Using Custom Services

Wakanda allows you to create and use custom **services** in your applications. Basically, a service is a piece of JavaScript code which makes a functionality available to the Wakanda application. For example, a "mail" service could provide functions to send or get emails.

Once launched, a service can communicate with the application through messages such as "applicationWillStop", allowing the JavaScript code to execute appropriate actions. Services can be started and stopped at any time, for example depending on specific events.

In Wakanda, a service is based on two parts:

- a CommonJS module,
- settings to define in the **.waSettings** file of the application.

A service can also embed some JavaScript files.

Wakanda provides you with a **Services** SSJS utility module containing several functions that can make services management easier.

Note: *For internal needs, Wakanda uses the following services:*

- *Datastore service*
- *RPC service*
- *WebApp service*

Building a CommonJS Module for Service

Wakanda service JavaScript code must be provided as a **CommonJS** module. For more information about CommonJS architecture, please refer to the CommonJS specification.

A Wakanda service CommonJS module must export a *postMessage* function:

```
    exports.postMessage = function( message) {
                // messages processing
    }
```

The application will use this function to notify the service when some events occur. The **message** object parameter contains at least one 'name' property and other properties required by the message.

The service can receive the following message names:

Message	Code	Comment
applicationWillStart	message.name = 'applicationWillStart'	The project is being started. It's the first message sent to the service
applicationWillStop	message.name = 'applicationWillStop'	The project is being closed
httpServerDidStart	message.name = 'httpServerDidStart'	The Wakanda HTTP Server is being started
httpServerWillStop	message.name = 'httpServerWillStop'	The Wakanda HTTP Server is being closed
catalogWillReload	message.name = 'catalogWillReload'	The datastore model is being reloaded on the server
catalogDidReload	message.name = 'catalogDidReload'	The datastore model has been reloaded on the server

The service module file is declared through the 'modulePath' setting (see **Defining the Settings for a Service**). This setting is an *id*, as defined in the CommonJS specification, just like the *id* parameter passed to the **require( )** method. It is usually a reference to a file located in the 'Modules' folder of the project. For example, if your 'modulePath' setting is "Modules/services/myService", you can access the module object using the following statement:

```
var myModule = require( 'services/myService')
```

How an Application Handles Services

First, the Wakanda application registers a service for each service setting found (see **Registering a Service**).

Then, once started, the application posts the 'applicationWillStart' message to each service:

```
message.name='applicationWillStart'
```

Note: *The 'applicationWillStart' message is posted before executing the application bootStrap.*

Calling an Application Service from Another One

By default, a service will work with the 'application' object which references the global object.

Therefore, an application may need access to the services of another application. In order to support this feature, the service must implement a function which will return a service instance of the target application. Take a look at the the **getInstanceFor( )** function code in the **Services** SSJS utility module to know how to implement such a function.

Defining the Settings for a Service

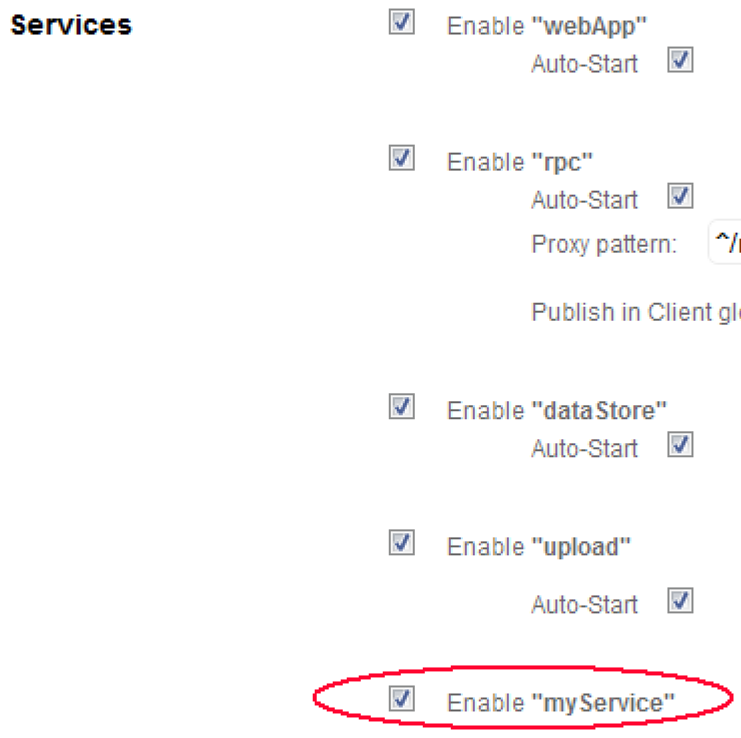
A service must be defined in the settings file of the project. The settings file is named ***{projectname}*.waSettings** and is located at the first level of the project folder. For more information, please refer to the **Project Settings Graphical Interface** paragraph.

To declare and configure a service, you have to add the following information in the settings file:

```
<service name="myService" modulePath="moduleID" enabled="true" [other attributes]/>
```

- *modulePath* (mandatory): ID of the CommonJS module. It should be the same value as the *id* parameter passed to the **require( )** function.
- *name* (optional but recommended): name used to add the service in the **application.settings.services** memory *Storage* object. If the name is not passed, Wakanda Server will try to extract it from the module ID.
- *enabled* (optional): to enable or disable the service; default value is "true"

Once declared, the service can be enabled/disabled from the Settings graphical interface in the Wakanda Studio:



Services

✓ Enable "webApp"
Auto-Start ✓

✓ Enable "rpc"
Auto-Start ✓
Proxy pattern: ^/rpc-proxy/
Publish in Client global Namespace ☐

✓ Enable "dataStore"
Auto-Start ✓

✓ Enable "upload"
Auto-Start ✓

✓ Enable "myService"

Accessing Settings

You can add any custom settings using attributes:

```
<service name="myService" modulePath="moduleID" enabled="true" autostart="true" optimizat
```

All service settings are automatically available through the **application.settings.services** memory *Storage* object:

```
var sName = settings.services.myService.name // contains "myService"
var sPath = settings.services.myService.modulePath // contains "moduleID"
var sAuto = settings.services.myService.autostart // contains "true"
var sOpt = settings.services.myService.optimization // contains "on"
```

Registering a Service

To be recognized by the application, each service must be registered. If you declared the service in the settings file (see above), the registration is automatic.

Otherwise, the service must be registered using the **registerService( )** utility function (see **Services** module description).

Once registered, the service has an entry in the **application.storage.services** memory *Storage* object:

```
var sName = storage.services.myService.name  // contains "myService"
var sPath = storage.services.myService.modulePath // contains "moduleID"
```

The service is also able to append its own data in the storage object, for example:

```
storage.services.myService.source="provided by Wakanda Team"
```

Detailed Example

In this example, we will implement a custom service named "soap", entirely written in JavaScript.

Settings File

In the settings file of the project (*myProject.waSettings*), we write:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
    ...
    service name="soap" modulePath="services/soap" enabled="true" autostart="true"
    ...
</settings>
```

The service module, named "soap.js", is located in the "Modules/services" folder of the project.

CommonJS Module

The module code is:

```
  /*    Constants for service state    */
    var kSTATE_STARTED    = 1;
    var kSTATE_STOPPED    = 2;
    var kSTATE_PAUSED    = 3;

    var kSERVICE_NAME    = 'soap';


    var servicesModule = require( 'services');


    /*    Soap Service implementation class
    */
    function soapImpl() {

        /*    Private members
        */

        /*    Initialize the instance
        */
        this._init = function () {
            this._context = {};
        }

        /*    Load the context of the service from the 'storage' object
        */
        this._loadContext = function () {

            var done = false;
            if (servicesModule.isServiceRegistered( kSERVICE_NAME)) {
                var servicesData = storage.getItem( 'services');
                this._context = servicesData[kSERVICE_NAME];
                if (!this._context.hasOwnProperty( 'state')) {
                    /* Initialize the service */
                    this._context.state = kSTATE_STOPPED;
                    /* Save the context */
                    servicesData[kSERVICE_NAME] = this._context;
                    storage.setItem('services', servicesData);
                }
                done = true;
            }

            return done;
        };

        /*    Save the context of the service in the 'storage' object
        */
        this._saveContext = function () {
```

```
    var done = false;
    if (servicesModule.isServiceRegistered( kSERVICE_NAME)) {
        var servicesData = storage.getItem( 'services');
        servicesData[kSERVICE_NAME] = this._context;
        storage.setItem( 'services', servicesData);
        done = true;
    }

    return done;
};

/*    Public members
*/

this.isStarted = function () {

    if (this._loadContext()) {
        return (this._context.state == kSTATE_STARTED);
    }
    return false;
};

/*    Start a stopped service
*/
this.start = function () {

    if (this._loadContext()) {
        if (this._context.state == kSTATE_STOPPED) {
            /* Install the soap request handler */
            this._context.state = kSTATE_STARTED;
            this._saveContext();
        }
    }
};

/*    Stop a started or paused service
*/
this.stop = function () {

    if (this._loadContext()) {
        if ((this._context.state == kSTATE_STARTED) || (this._context.state == kS
            /* Uninstall the soap request handler */
            this._context.state = kSTATE_STOPPED;
            this._saveContext();
        }
    }
};

/*    Pause a started service
*/
this.pause = function () {

    if (this._loadContext()) {
        if (this._context.state == kSTATE_STARTED) {
            /* Uninstall the soap request handler */
            this._context.state = kSTATE_PAUSED;
            this._saveContext();
        }
    }
};

/*    Resume a paused service
*/
this.resume = function () {

    if (this._loadContext()) {
        if (this._context.state == kSTATE_PAUSED) {
            /* Install the soap request handler */
```

```
                this._context.state = kSTATE_STARTED;
                this._saveContext();
            }
        }
    };

    /*    Initialize the instance
    */
    this._init();

    return this;
}


/*    Create service implementation
*/
var impl = new soapImpl();


/*    Handler for service messages
*/
exports.postMessage = function (message) {

    if ((impl != null) && (typeof impl != 'undefined')) {

        if (message.name === "applicationWillStart") {
            var serviceSettings = settings.getItem('services');
            if (serviceSettings[kSERVICE_NAME].hasOwnProperty( 'autoStart')) {
                if (serviceSettings[kSERVICE_NAME].autoStart === "true") {
                    impl.start();
                }
            }
        }
        else if (message.name === "applicationWillStop") {
            impl.stop();
        }
        else if (message.name === "httpServerWillStop") {
            impl.pause();
        }
        else if (message.name === "httpServerDidStart") {
            impl.resume();
        }
    }
};


exports.start = function () {

    if ((impl != null) && (typeof impl != 'undefined')) {
        impl.start();
    }
};


exports.stop = function () {

    if ((impl != null) && (typeof impl != 'undefined')) {
        impl.stop();
    }
};


exports.isStarted = function () {

    if ((impl != null) && (typeof impl != 'undefined')) {
        return impl.isStarted();
    }
    return false;
};
```