

```
/**
 * Backend class
 * modified: 02/11/2007
 * copyright: muevo Information Management
 * contact: office@muevo.at, www.muevo.at
 *
 * This class capsulates all the backend logic (SWX). The Singleton design pattern is used to make sure
 * that there exists only one instance of this class throughout the whole application.
 */
// import necessary classes
import at.muevo.tonis.suche.*;
import org.swxformat.*;

class at.muevo.tonis.suche.Backend {

    // definition of path to configuration file
    private static var _configFile:String = 'config.xml';

    // definition of parameter name for debug flag
    private static var _configParameterDebug:String = 'debug_mode';

    // definition of parameter name for backend domain (security)
    private static var _configParameterDomain:String = 'backend_domain';

    // definition of parameter name for backend gateway
    private static var _configParameterGateway:String = 'backend_gateway';

    // definition of parameter name for the images
    private static var _configParameterImagePath:String = 'image_path';

    // private static instance variable holding the Singleton instance
    private static var _instance:Backend = null;

    // private variable indicating if Backend is initialized
    private var _isInitialized:Boolean = false;

    private var _isConfigLoaded:Boolean = false;

    // array holding the externally loaded configuration parameters
    public var configParameters:Array = new Array();

    // private instance variable holding the SWX object
    private var swx:SWX = null;

    // holds the value of the last SWX query
    private var queryResult:Object = null;

    // flag indicating if a query is running
    private var queryIsRunning:Boolean = false;

    // flag indicating if the last query timed out
    private var queryhasTimedOut:Boolean = false;

    // flag indicating if the last query failed
    private var queryhasError:Boolean = false;

    // holds the current query progress
```

```

private var queryProgress;

//to get the details out of the object
private var bauernInfos:Object = null;

// *****
// public methods (API)
// *****

/**
 * This method gets the path to the images - either detail (1) or normal (0)
 */
public function getImagePath(p_id:String, p_getDetail:Boolean):String{
    if (p_getDetail){
        return this.configParameters[Backend._configParameterImagePath] + "detail/" +
p_id;
    }
    else {
        return this.configParameters[Backend._configParameterImagePath] + "photo/" + p_id;
    }
}

//return Data for clicked bauer
public function getDetailsForBauer (p_id:String) :Array{
    //trace("GET DETAILS FOR BAUER" + bauernInfos[p_id]);
    return bauernInfos [p_id];
}

/**
 * This method triggers a simple SWX connection check
 */
public function checkConnection():Void {
    var callDetails:Object =
    {
        serviceClass:"SwxController",
        method:"test",
        result:[this, _connectionCheckResultCallback],
        fault: [this, _swxFaultCallback],
        timeout: [this, _swxTimeoutCallback],
        progress: [this, _swxProgressCallback]
    };

    // call backend
    this.doSWXCall(callDetails);
}

/**
 * This method triggers the loading of the complete data set from SWX backend
 */
public function loadData():Void {
    var callDetails:Object =
    {
        serviceClass:"SwxController",
        method:"loadAllData",
        result:[this, _loadDataResultCallback],
        fault: [this, _swxFaultCallback],
        timeout: [this, _swxTimeoutCallback],
        progress: [this, _swxProgressCallback]
    }
}

```

```
    };
    // call backend
    this.doSWXCall (callDetails);
}

/**
 * This method triggers a query for IDs for initial (random) farmers
 */
public function getInitIDs():Void {
    var callDetails:Object =
        {
            serviceClass:"SwxController",
            method:"findInit",
            result:[this, _getInitIDsResultCallback],
            fault: [this, _swxFaultCallback],
            timeout: [this, _swxTimeoutCallback],
            progress: [this, _swxProgressCallback]
        };
    // call backend
    this.doSWXCall (callDetails);
}

/**
 * This method triggers a query for IDs by name
 */
public function getIDsByName (p_name:String):Void {
    var callDetails:Object =
        {
            serviceClass:"SwxController",
            method:"findName",
            args: [p_name],
            result:[this, _getIDsByNameResultCallback],
            fault: [this, _swxFaultCallback],
            timeout: [this, _swxTimeoutCallback],
            progress: [this, _swxProgressCallback]
        };
    // call backend
    this.doSWXCall (callDetails);
}

/**
 * This method triggers a query for IDs by ZIP code
 */
public function getIDsByZIP (p_zip:String):Void {
    var callDetails:Object =
        {
            serviceClass:"SwxController",
            method:"findZip",
            args: [p_zip],
            result:[this, _getIDsByZIPResultCallback],
            fault: [this, _swxFaultCallback],
            timeout: [this, _swxTimeoutCallback],
            progress: [this, _swxProgressCallback]
        };
    // call backend
    this.doSWXCall (callDetails);
}

/**
 * This method triggers a query for IDs by control number
```

```

*/
public function getIDsByEi (p_ei:String):Void {
    var callDetails:Object =
        {   serviceClass:"SwxController",
            method:"findEiNummer",
            args: [p_ei],
            result:[this, _getIDsByEiResultCallback],
            fault: [this, _swxFaultCallback],
            timeout: [this, _swxTimeoutCallback],
            progress: [this, _swxProgressCallback]
        };
    // call backend
    this.doSWXCall (callDetails);
}

// *****
// constructor
// *****

/**
 * This is the singleton constructor, returning the only existing instance of the object
 */
public static function getInstance():Backend{
    // if no instance exists yet, create new one
    if (Backend._instance == null) {
        Backend._instance = new Backend();
    }
    // return instance
    return Backend._instance;
}

/**
 * The constructor should be private (Singleton), but AS does not allow this.
 * It should never be called from outside this class!
 */
function Backend() {
    // initialize
    trace("start init of backend");

    this._init();
}

// *****
// private methods
// *****

/**
 * This private method is used to initialize the Backend.
 * The instance flag _initialized is used to make sure that backend is not initialized multiple times
 */
private function _init():Void {
    // trigger loading of the configuration
    this._loadConfiguration();
}

/**
 * This private method loads an external configuration file defined by Backend._configFile.

```

```
* The parameters are stored in an array, so that they can be made accessible
*/
private function _loadConfiguration():Void {
    // store object reference for use in onLoad
    var thisObj = this;
    trace("loading configuration");

    // prepare xml
    var config = new XML();
    config.ignoreWhite = true;

    // define handler function
    config.onLoad = function(success:Boolean):Void {
        if(success) {
            trace("config parsed");
            // go through all parameters in config file
            for (var i = this.childNodes[0].firstChild; i != null; i=i.nextSibling) {
                // convert to boolean if necessary
                if(i.attributes.value == String(true)) { // set boolean true
                    thisObj.configParameters[i.attributes.key] = true;
                }
                else if (i.attributes.value == String(false)) { // set boolean false
                    thisObj.configParameters[i.attributes.key] = false;
                }
                else { // set plain string value
                    thisObj.configParameters[i.attributes.key] = [i.attributes.value];
                }
            }

            // finish intialization
            thisObj._completeInitialization();
        }
        else {
            trace("Error reading configuration.");
        }
    }

    // load configuration
    config.load(Backend._configFile);
}

/**
 * This method finishes the backend initialization after all necessary configuration paramters have
 * been loaded (callback)
 */
private function _completeInitialization() {

    // set properties
    System.useCodepage = true;
    System.security.allowDomain(this.configParameters[Backend._configParameterDomain]);

    // setup SWX backend
    this.swx = new SWX();
    this.swx.gateway = this.configParameters[Backend._configParameterGateway];
    this.swx.encoding = 'POST';
    this.swx.debug = this.configParameters[Backend._configParameterDebug];
}
```

```
// test connection
// this.checkConnection(); // we only used this for debugging

// set flag
this._isInitialized = true;

}

// *****
// SWX related functions
// *****

private function doSWXCall(p_details:Object):Void {
    // prepare flags
    this.queryResult = null;
    this.queryhasTimedOut = false;
    this.queryhasError = false;
    this.queryIsRunning = true;
    // do call
    this.swx.call(p_details);
}

/**
 * This is a generic SWX callback invoked by SWX backend on result
 */
public function _swxResultCallback(resultObj:Object):Void {
    trace("SWX call finished. number of results: " + resultObj.result.length);
    this.queryResult = resultObj;
    this.queryIsRunning = false;
}

/**
 * This is a SWX callback invoked by SWX backend in case of error
 */
public function _swxFaultCallback(resultObj:Object):Void {
    trace ("SWX call failed: " + resultObj.fault.message);
    this.queryhasError = true;
    this.queryIsRunning = false;
}

/**
 * This is a SWX callback invoked by SWX backend in case of a timeout
 */
public function _swxTimeoutCallback():Void {
    trace("SWX call timed out");
    this.queryhasTimedOut = true;
    this.queryIsRunning = false;
}

/**
 * This is a SWX callback invoked by SWX backend to give progress feedback
 */
public function _swxProgressCallback(resultObj:Object):Void {
    // trace("progress callback: " + resultObj);
    // trace("progress bytesLoaded: " + resultObj.bytesLoaded);
    // trace("progress bytesTotal: " + resultObj.bytesTotal);
}
}
```

```
/**
 * This is the result callback for the connection test
 */
public function _connectionCheckResultCallback(resultObj:Object):Void {
    // manually invoke generic callback
    this._swxResultCallback(resultObj);
    // do special handling
    if(resultObj.result){
        trace("SWX communication works");
    }
    else{
        trace("SWX communication NOT working");
    }
}

/**
 * This is the result handler for the loadData call
 */
public function _loadDataResultCallback(resultObj:Object):Void {
    // manually invoke generic callback
    this._swxResultCallback(resultObj);
    // do special handling
    // prepare bauern array
    //var tmpArray = new Object(); => bauernInfos
    bauernInfos = new Object();
    for (var i:Number = 0; i<resultObj.result.length; i++) {
        //trace("aktuelle ID: " + resultObj.result[i].Bauer.LfBisNr);
        bauernInfos[resultObj.result[i].Bauer.LfBisNr] = resultObj.result[i];
    }
    bauernInfos.anzahl = resultObj.result.length;
    // invoke data handler
    _global.bauernSuche._loadDataResultHandler(bauernInfos);
}

/**
 * This is the result handler for the getInitIDs call
 */
public function _getInitIDsResultCallback(resultObj:Object):Void {
    // manually invoke generic callback
    this._swxResultCallback(resultObj);
    // invoke data handler
    _global.bauernSuche._drawInitScreenResultHandler(resultObj.result);
}

/**
 * This is the result handler for the getIDsByName call
 */
public function _getIDsByNameResultCallback(resultObj:Object):Void {
    // manually invoke generic callback
    this._swxResultCallback(resultObj);
    // invoke data handler
    _global.bauernSuche._findByNameResultHandler(resultObj.result);
}

/**
 * This is the result handler for the getIDsByZIP call
```

```
*/
public function _getIDsByZIPResultCallback(resultObj:Object):Void {
    // manually invoke generic callback
    this._swxResultCallback(resultObj);
    // invoke data handler
    _global.bauernSuche._findByZIPResultHandler(resultObj.result);
}

/**
 * This is the result handler for the getIDsByEi call
 */
public function _getIDsByEiResultCallback(resultObj:Object):Void {
    // manually invoke generic callback
    this._swxResultCallback(resultObj);
    // invoke data handler
    _global.bauernSuche._findByNumberResultHandler(resultObj.result);
}
}
```