ARE RESTful API

Table of Contents

[RESTful API 2](#_Toc419980823)

[RESTful API libraries 4](#_Toc419980824)

[The JavaScript library 4](#_Toc419980825)

[The Java library 7](#_Toc419980826)

# RESTful API

To allow remote communication with the AsTeRICS Runtime Environment, the ARE RESTful API was developed. It allows manipulation of resources through a set of HTTP methods such as GET, POST, PUT and DELETE.

Apart from the regular RESTful functions, an event mechanism is provided. With this mechanism, ARE can broadcast messages to anyone who subscribes and inform when an event occurs.

The API uses HTTP status codes to declare an error in a call. Specifically, when an error occurs, the response will contain a 500 HTTP status code (Internal Server Error) with an ARE-produced error message inside the HTTP response body.

The figure in the next page describes these methods and provides the necessary information in order to call them.

RESTful API Functions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **HTTP Method** | **Resource** | **Parameters** | **Consumes** | **Produces** | **Description** |
| GET | /runtime/model | - | - | XML | Returns the currently deployed model in XML |
| PUT | /runtime/model | modelInXML  (in body) | XML | TEXT | Deploys the model given as a parameter |
| PUT | /runtime/model/{filename} | filename | - | TEXT | Deploys the model contained in the given filename |
| PUT | /runtime/model/state/{state} | state | - | TEXT | Changes the state of the deployed model to STARTED, PAUSED, STOPPED |
| GET | /runtime/model/state | - | - | TEXT | Returns the state of the deployed model |
| PUT | /runtime/model/autorun/  {filename} | filename | - | TEXT | Deploys and starts the model in the given filename |
| GET | /runtime/model/components | - | - | JSON | Returns all the components contained in the currently deployed model |
| GET | /runtime/model/components/  {componentId} | componentId | - | JSON | Returns all property keys of the component with the given componentId in the currently deployed model |
| GET | /runtime/model/components/  {componentId}/{componentKey} | componentId,  componentKey | - | TEXT | Returns property value of a specific component, in the currently deployed model |
| PUT | /runtime/model/components/  {componentId}/{componentKey} | componentId,  componentKey,  value (in body) | TEXT | TEXT | Changes a property value of a specific component, in the currently deployed model |
| GET | /storage/models/{filename} | filename | - | XML | Returns an xml representation of a model in a specific file |
| POST | /storage/models/{filename} | filename,  modelInXML  (in body) | XML | TEXT | Stores a model in the given filename |
| DELETE | /storage/models/{filename} | filename | - | TEXT | Deletes the model with the given filename |
| GET | /storage/models | - | - | JSON | Returns a list with all the models that are saved in the ARE repository |
| GET | /storage/components/installed | - | - | JSON | Returns an array containing the installed ARE components |
| GET | /storage/components/installed/ descriptors | - | - | XML | Returns an xml string containing the descriptors of the installed components |
| GET | /storage/components/created/ descriptors | - | - | XML | Returns an xml string containing the descriptors of the created components |
| GET | /restfunctions | - | - | JSON | Returns a list with all the available rest functions |
| GET | /events/subscribe | - | - | - | Opens a persistent connection with ARE to use it for Server Sent Events. |

# 

# RESTful API libraries

To provide easier RESTful API accessibility, communication libraries were created that simplify the whole procedure.

## The JavaScript library

To install the JavaScript library in a webpage you have to:

1. Import the ‘ARECommunicator.js’ file in your html page.
2. Import ‘JSmap.js’ file in your html page.
3. Import a script that provides jQuery functionality.

Before you start calling ARE functions, you have to set the baseURI which is the URI where ARE runs at:

setBaseURI("http://localhost:8081/rest");

//downloadDeployedModel

function DDM() {
     downloadDeployedModel(DDM_successCallback, DDM_errorCallback);
}
   
function DDM_successCallback(data, HTTPstatus) {
     alert(data);
}
   
function DDM_errorCallback(HTTPstatus, AREerrorMessage) {
     alert(AREerrorMessage);
}To call any REST function, you have to provide two callback functions: a successCallback and an errorCallback such as the example below

Furthermore, the ‘subscribe’ function is opening a persistent connection with ARE. Using an event mechanism based on Server Sent Events (SSE) technology, it listens to the connection for broadcasted messages. Additionally, the eventType name must be provided, to specify what type of events to listen for. The concept still remains the same as you must provide a successCallback and an errorCallback function. The unsubscribe function do not use any rest calls since it closes the connection from the browser’s side.

In below you can find an array describing each method contained in the library and a listing with the available event types.

|  |  |
| --- | --- |
| **Function Signature** | **Description** |
| downloadDeployedModel(**sCB1,**  eCB) | Retrieves the currently deployed model in XML |
| uploadModel(**sCB1,**  eCB**,** modelinXML) | Deploys the model given as a parameter |
| deployModelFromFile(**sCB1,**  eCB**,** filename) | Deploys the model contained in the given filename |
| startModel(**sCB1,**  eCB)  stopModel(**sCB1,**  eCB)  pauseMolel(**sCB1,**  eCB) | Changes the state of the deployed model to STARTED, PAUSED, STOPPED |
| getModelState( **sCB1,**  eCB) | Retrieves the state of the deployed model |
| autorun(**CB1,**  eCB**,** filename) | Deploys and starts the model in the given filename |
| downloadComponentCollection(**sCB1,**  eCB) | Retrieves all the components contained in the currently deployed model (as JSON array) |
| getComponentPropertyKeys(**sCB2,**  eCB**,**  componentId) | Retrieves all property keys of the component with the given componentId in the currently deployed model (as JSON array) |
| getComponentProperty(**sCB1,**  eCB**,**  componentId, componentKey) | Retrieves property value of a specific component, in the currently deployed model |
| setComponentProperty(**sCB1,**  eCB**,**  componentId, componentKey, value) | Changes a property value of a specific component, in the currently deployed model |
| downloadModelFromFile(**sCB1,**  eCB**,** filename) | Retrieves an xml representation of a model in a specific file |
| storeModel(**sCB1,**  eCB**,** filename**,** modelinXML) | Stores a model in the given filename |
| deleteModelFromFile(**sCB1,**  eCB**,** filename) | Deletes the model with the given filename |
| listStoredModels(**sCB2,**  eCB) | Retrieves the models that are saved in the ARE repository (as JSON array) |
| getInstalledComponents(**sCB2,**  eCB) \*\* | Retrieves the components which are currently installed in the ARE (as JSON array with Component objects) |
| getInstalledComponentsDescriptor(**sCB2,**  eCB) | Returns an xml string containing the descriptors of the installed components |
| getCreatedComponentsDescriptor(**sCB2,**  eCB) | Returns an xml string containing the descriptors of the created components |
| getRestFunctions(**sCB2,**  eCB) \*\*\* | Retrieves the information for all the available rest functions provided by the Restful API (as JSON array with Function objects) |
| subscribe(**sCB1,**  eCB, eventType) | Opens a persistent connection with ARE and listens for Server Sent Events. |
| unsubscribe(eventType) | Closes the connection for Server Sent Events. Returns true if the unsubscription was successful and false otherwise |

JavaScript Library Functions

**sCB1**: successCallback(textData, HTTPstatus)

**sCB2**: successCallback(array, HTTPstatus)

eCB: errorCallback(HTTPstatus, AREerrorMessage)

\*\*: Component object (see JSON objects section)

\*\*\*: Function object (see JSON objects section)

Event Types

|  |  |
| --- | --- |
| **Event Type Name** | **Description** |
| Model State Changed | Notifies the subscribers that model state was changed (started, stopped, paused) |
| Model changed | Notifies the subscribers that model was changed |
| Repository changed \* | Notifies the subscribers the ARE repository content was changed |

\* NOT YET IMPLEMENTED

JSON OBJECTS

|  |  |
| --- | --- |
| **Object Name** | **Example** |
| Function | {  "path": "/runtime/model",  "description": "Retrieves the currently deployed model in XML",  "httpRequestType": "GET",  "bodyParameter": "",  "consumes": "",  "produces": "text/xml"  } |
| Component | {  "canonicalName":"eu.asterics.component.processor….",  "type":"PROCESSOR",  "id":"asterics.StringDispatcher",  "description":"Send text from chosen slot",  "singleton":false,  "inputPorts":[  {  "type":"INPUT",  "multiplicity":null,  "description":"Send the string from the slot defined by the incoming value",  "portID":"slotDispatch",  "dataType":"INTEGER",  "propertyNames":null  }  ],  "outputPorts":[  {  "type":"OUTPUT",  "description":"Output text",  "portID":"output",  "dataType":"STRING",  "propertyNames":null  }  ],  "eventTriggererPorts":[  ],  "ports":[  {  "type":"INPUT",  "multiplicity":null,  "description":"Send the string from the slot defined by the incoming value",  "portID":"slotDispatch",  "dataType":"INTEGER",  "propertyNames":null  },  {  "type":"OUTPUT",  "description":"Output text",  "portID":"output",  "dataType":"STRING",  "propertyNames":null  }  ],  "eventPorts":[  {  "id":"dispatchSlot1",  "description":"Send text from slot 1"  }  ],  "eventListenerPorts":[  {  "id":"dispatchSlot1",  "description":"Send text from slot 1"  }  ],  "propertyNames":[  "delay",  "slot1"  ]  } |

## The Java library

To use the JAVA framework in your code, you have to import the ‘ARECommunicator.jar’ file. When you do this, the procedure of communicating with ARE is reduced in plain calls of Java methods of an object.

As with JavaScript framework, you must first set the baseURI:

when this is done, you can call any method:



In the next page you can find an array that describes each method that is contained in the library.

|  |  |
| --- | --- |
| **Function Signature** | **Description** |
| **String** downloadDeployedModel() | Retrieves the currently deployed model in XML |
| **String** uploadModel(**String** modelinXML) | Deploys the model given as a parameter |
| **String** deployModelFromFile(**String** filename) | Deploys the model contained in the given filename |
| **String** startModel()  **String** stopModel()  **String** pauseModel(**)** | Changes the state of the deployed model to STARTED, PAUSED, STOPPED |
| **String** getModelState() | Retrieves the state of the deployed model |
| **String** autorun(**String** filename) | Deploys and starts the model in the given filename |
| **String[]** downloadComponentCollection() | Retrieves all the components contained in the currently deployed model |
| **String[]** getComponentPropertyKeys(**String** componentId) | Retrieves all property keys of the component with the given componentId in the currently deployed model |
| **String** getComponentProperty(**String** componentId, **String** componentKey) | Retrieves property value of a specific component, in the currently deployed model |
| **String** setComponentProperty(**String** componentId, **String** componentKey, **String** value) | Changes a property value of a specific component, in the currently deployed model |
| **String** downloadModelFromFile(**String** filename) | Retrieves an xml representation of a model in a specific file |
| **String** storeModel(**String** filename**, String** modelinXML) | Stores a model in the given filename |
| **String** deleteModelFromFile(**String** filename) | Deletes the model with the given filename |
| **String[]** listStoredModels() | Retrieves a list with all the model that are saved in the ARE repository |
| **String[]** getInstalledComponents() \* | Returns a list containing all the available ARE components |
| **String** getInstalledComponentsDescriptor() \* |  |
| **String** getCreatedComponentsDescriptor() \* |  |
| **ArrayList<RestFunction>** functions() | Retrieves a list with all the available rest functions |
| subscribe(**String** eventType) \* | Subscribes the IP that sent the request to the event mechanism |
| unsubscribe(**String** eventType) \* | Unsubscribes the IP that sent the request to the event mechanism |

Java Library Methods

\* NOT YET IMPLEMENTED