Accessing Caché Source Code Files Using REST

Version 2018.1
2018-12-13

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Accessing Caché Source Code Files Using REST
Caché Version 2018.1 2018-12-13

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book describes how to access Caché source code files, which include class definitions, routines, and CSP files, using REST. This API was created to allow the Atelier development environment to access Caché source code files. You can use this REST API to implement a development environment, a Caché class explorer, or a similar application.

The book contains the following topics:

- Introduction to the Caché Source Code File REST API
- Caché Source Code File REST API Tutorial
- Caché Source Code File REST API Reference

This document describes the API used to implement the Atelier development environment, which is an add-on for Eclipse. If you want information about using Atelier to develop Caché applications, see http://www.intersystems.com/atelier, the Atelier home page.

For general information, see *Using InterSystems Documentation*.

1

Introduction to the Caché Source Code File REST API

This REST API provides access to Caché source code files. We developed this API to allow Atelier, our Eclipse-based IDE, to access Caché class definitions, routines, and CSP files on a Caché server. You can use the API to perform the actions needed to access Caché classes, routines, and CSP files. These actions include:

- Getting the namespaces available on a Caché instance
- Finding the class definitions, routines, and CSP files defined in a namespace
- Getting the text definitions of the classes, routines, and CSP files
- Updating a class definition, routine, or CSP file
- Creating a new class definition, routine, or CSP file
- Deleting a class definition, routine, or CSP file
- Compiling a Caché class or routine
- Discovering properties of the Caché environment by performing SQL queries on tables

These actions provide the mechanisms to access Caché source code files. In order to create a Caché development environment, you should understand this API and have a comprehensive understanding of how the Caché source code files are used within Caché.

This is a special-purpose API. If you are creating a development environment or working on a similar application, such as a class browser, you may find this API useful. But, it is not a general-purpose REST API to access Caché objects.

If you want information about downloading Atelier and about using Atelier to develop Caché applications, see http://www.intersystems.com/atelier, the Atelier home page, which provides links to downloading instructions and to the Atelier documentation.

This document describes versions 1 and 2 of the Caché Source Code File REST API. Version 1 was first provided by Caché 2016.2 and version 2 by Caché 2017.2. Future releases of Caché may support higher versions of this REST API that provide additional calls, but you will always be able to call the earlier versions. The APIs for version 1 include `/v1/` in the URL and the APIs for version 2 include `/v2/`. You can find out the version of this API that is provided by Caché by calling the GetServer method.

The following describes the major capabilities of the API and the methods that provide them:

- Getting information about the server environment:
  - GetServer method provides the important information about the server including the namespaces on the server.

  - GetNamespace method provides additional information about the specified namespace. It includes the list of databases that are mapped to the namespace.
  - HeadServer method provides header information about the server. You can call HeadServer to check if the server is available.
  - GetJobs method provides information about the jobs running in Caché.
  - GetCSPApps method provides information about the CSP applications. These applications provide access to Caché.

- Getting information about source code files:
  - GetDocNames method provides the names of the source code files in the namespace. You can optionally limit the files to a specific category of files or with a specific file type.
  - GetModifiedDocNames method provides the same names as GetDocNames, but additionally provides a hash for the database state. If you keep a local copy of the file, you can call GetModifiedDocNames and see if the document has changed since you last retrieved it.
  - GetDoc method gets the contents of the specified source code file. You can optionally use the ETAG and If-None-Match headers to only get the contents of the source code file if it has changed since the previous time you retrieved it.
  - GetDocs method gets the contents of the specified files.
  - Index method provides some key properties of the class definitions in the namespace. Your application can use this information to select the class definitions to access.
  - HeadDoc method provides header information about source code files.

- Creating, updating, and deleting source code files
  - PutDoc method updates an existing source code file, or, if the file does not exist, creates a new source code file.
  - DeleteDoc method deletes the specified source code file.
  - DeleteDocs method deletes the specified list of source code files.

- Compiling source code files
  - Compile method compiles the source code file.

- Performing SQL queries to get information from Caché tables
  - Query method performs an SQL query on any Caché database.

- Searching in source code files
  - Search method searches source code files in the Caché database.

- Special calls for dealing with Ensemble classes
  - GetEnsClassType method returns the class type for Ensemble objects.
  - GetAdpInputOutputClass method returns the input and output adapter class for Ensemble adapters.

2

Caché Source Code File REST API Tutorial

This chapter provides a brief tutorial that demonstrates how to use the Caché Source Code File REST API by a series of examples. It contains the following sections:

2.1 API Basics

The API used by Atelier to access Caché source code files uses the REST architectural style. REST is named from "Representational State Transfer." The Caché Source Code File REST API, like many REST APIs, uses the HTTP GET, POST, PUT, DELETE, and HEAD methods and uses JSON for incoming and outgoing message bodies.

To call an API method, you need to know the following:

- HTTP method—which is one of the following: GET, POST, PUT, DELETE, or HEAD.
- HTTP headers—provide context information for the call. The HEADERS used in this API include:
  - Authorization, which provides access to the server. Unless you have installed your server with minimal security, you need to provide a username and password to access the API.
  - `Content-Type application/json`, which specifies that the inbound payload is provided in JSON. You must specify this header for all POST and PUT methods.
  - `If-None-Match`, which allows a GetDoc or PutDoc call to check if the source code file was modified since it was last accessed.
- URL—The URL consists of the following parts:
  - `http://`

  - *server-name*:*port-number*/—In this chapter, we assume that Caché is running on the local server and is using port 57772.
  - `api/atelier/`—This is defined by the web application that has the %Api.Atelier dispatch class.
  - URL part that identifies the method and target. This part can include fixed text and text that you specify to identify the namespace, document name, or type.

  For example, the URL part that identifies the GetDocNames method is `v1/`*namespace*`/docs/`. The complete URL for this method getting the documents from the SAMPLES namespace would be:

  ```
  http://localhost:57772/api/atelier/v1/SAMPLES/docnames
  ```

  The URL part that identifies the GetServer method is an empty string, so the complete URL for GetServer is:

  ```
  http://localhost:57772/api/atelier/
  ```

- URL parameters—modifies the call. If the API method has URL parameters, they are described in the reference section.
- Inbound JSON payload—format of the inbound message for POST and PUT methods.
- Outbound JSON payload—format of the outbound message returned by the HTTP method.

2.2 Getting Information about the Caché Server

Typically, the first REST call you'll make is to the GetServer method, which returns information about the Caché Source Code File REST API version number and the namespaces available on the server.

```
GET http://localhost:57772/api/atelier/
```

This call returns the following JSON message:


```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": {
      "version": "Cache for Windows (x86-64) 2017.2 (Build 691U) Wed Jun 7 2017 20:45:20 EDT",
      "id": "B70A630D-A34D-43B1-8EA5-EDF8F38992C4",
      "api": 2,
      "features": [
        {
          "name": "DEEPSEE",
          "enabled": true
        },
        {
          "name": "ENSEMBLE",
          "enabled": true
        },
        {
          "name": "HEALTHSHARE",
          "enabled": false
        }
      ],
      "namespaces": [
        "%SYS",
        "DOCBOOK",
        "ENSDEMO",
        "ENSEMBLE",
        "INVENTORY",
        "SAMPLES",
        "USER"
      ]
    }
  }
}
```

All Caché Source Code File REST API methods that return JSON messages use the same general format:

- status errors—typically the Caché Source Code File REST API returns errors as HTTP status codes. This field is used under some unusual conditions and this element contains the Caché %Status value, which may contain the text for multiple errors.
- status summary—contains a summary of the status errors.
- console—contains the text that Caché would display on the console for this operation.
- result—contains the results of the method.

The GetServer method returns information about the server in the “result” element. The result element contains one value “content”, which contains:

- version—contains the version string of the instance of Caché or Ensemble running on the server.
- id—contains the instance GUID of Caché.
- api—specifies the version number of the Caché Source Code File REST API implemented in this version of Caché. In Caché 2016.2 and 2017.1, this version is 1. In Caché 2017.2, this version is 2. If the version returned in this element is 3 or greater, then there will be additional APIs that you can call.
- features—indicates whether features such as DEEPSEE, ENSEMBLE, or HEALTHSHARE are enabled on this instance.
- namespaces—lists the namespaces defined on the Caché server.

The GetNamespace method returns information about the specified namespace, including the databases that are mapped to the namespace and a hash for each database. The hash is useful for improving efficiency of communication with the server. But you can get the information about the source code files in the namespace with just the namespace information returned by GetServer.

2.3 Getting the Source Code Files Defined in a Namespace

To get the information about the source code files in a namespace:

- First you get the names of the files with the GetDocNames method.
- Then you get the contents of one file with the GetDoc method or you can get the contents of multiple files with the GetDocs method.
- If you want to improve the network efficiency of your application, you can keep a local cache of the names and contents of the source code files and use the GetModifiedDocNames method to get only the names of the source code files whose contents have changed or use the GetDoc method with the `If-None-Match` HTTP header.

The GetDocNames method returns the names of all of the source code files in all databases mapped to the namespace.

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": [
      {
        "name": "%Activate.Enum.cls",
        "cat": "CLS",
        "ts": "2016-08-03 20:01:42.000",
        "upd": true,
        "db": "CACHELIB",
        "gen": false
      },
```

```
      ...
      {
        "name": "EnsProfile.mac",
        "cat": "RTN",
        "ts": "2003-09-19 13:53:31.000",
        "upd": true,
        "db": "INVENTORYR",
        "gen": false
      },
      ...
      {
        "name": "xyz.mac",
        "cat": "RTN",
        "ts": "2016-08-11 15:05:02.167",
        "upd": false,
        "db": "INVENTORYR",
        "gen": false
      }
    ]
  }
}
```

The following GetDoc call returns the contents of the xyz.mac file:

```
http://localhost:57772/api/atelier/v1/INVENTORY/doc/xyz.mac
```

This call returns:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "name": "xyz.mac",
    "db": "INVENTORYR",
    "ts": "2016-09-14 14:10:16.540",
    "upd": false,
    "cat": "RTN",
    "status": "",
    "enc": false,
    "flags": 0,
    "content": [
      "ROUTINE xyz",
      "xyz ;",
      "    w \"hello\""
    ]
  }
}
```

2.4 Creating a New File in a Namespace or Updating an Existing File

To create a new file in a namespace or update an existing file, you use the PutDoc method. For example, the following REST call creates a new xyz.mac source code file in the INVENTORY namespace or, if the xyz.mac file exists, this call replaces the original definition of the file with the new one. If you are updating a new file, you must specify either the HTTP header `If-None-Match` to identify the current version of the file or the `?ignoreConflict=1` URL parameter to bypass version checking. See PutDoc in the reference section for details.

```
PUT http://localhost:57772/api/atelier/v1/INVENTORY/doc/xyz.mac
```

You should specify the Content-Type application/json and the following JSON message:

```
{
 "enc": false,
 "content": [
   "ROUTINE xyz",
   "xyz ;",
   "   w \"hello\""
   ]
}
```

The call returns the following JSON message. It shows that the source code file has been created in the INVENTORYR database, which is the default database for routines in the INVENTORY namespace.

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "name": "xyz.mac",
    "db": "INVENTORYR",
    "ts": "2016-09-14 14:10:16.540",
    "upd": false,
    "cat": "RTN",
    "status": "",
    "enc": false,
    "flags": 0,
    "content": []
  }
}
```

If you are updating or creating a binary file, specify a true value for `enc` and include the binary contents as an array of blocks of the base64 encoding of the binary value.

2.5 Compiling a File

The Compile method compiles the source code files specified by name in the incoming JSON array. For example, to compile xyz.mac, POST the following:

```
http://localhost:57772/api/atelier/v1/INVENTORY/action/compile
```

with the following JSON message:

```
["xyz.mac"]
```

The method returns:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [
    "",
    "Compilation started on 08/14/2016 15:25:20 with qualifiers 'cuk'",
    "xyz.int is up to date. Compile of this item skipped.",
    "Compilation finished successfully in 0.008s."
  ],
  "result": {
    "content": []
  }
}
```

For some source code files, such as classes, Compile returns storage information in the returned content.

2.6 Deleting a File

The DeleteDoc method deletes the file specified in the URL. The DeleteDoc method has the same URL as the GetDoc method except that you use the HTTP Delete method instead of the Get Method. To delete xyz.mac, make an HTTP DELETE request with the URL:

```
http://localhost:57772/api/atelier/v1/INVENTORY/doc/xyz.mac
```

The Delete method returns the following JSON message:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "name": "xyz.mac",
    "db": "INVENTORYR",
    "ts": "",
    "cat": "RTN",
    "status": "",
    "enc": false,
    "flags": 0,
    "content": []
  }
}
```

When a file has been deleted, the timestamp, `ts`, has a value of "" (empty string).

2.7 Performing an SQL Query

The Query method performs an SQL query on any Caché database. For example, if your application wants to present the user with a list of Caché roles, it can discover them with the following call:

```
POST http://localhost:57772/api/atelier/v1/%25SYS/action/query
```

With the SQL query specified in the incoming JSON message:

```
{"query": "SELECT ID,Description FROM Security.Roles"}
```

This call returns the results of the SQL query as JSON in the result content element.

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": [
      {
        "ID": "%all",
        "Description": "The Super-User Role"
      },
      {
        "ID": "%db_%default",
        "Description": "R/W access for this resource"
      },
      ...
      {
        "ID": "%sqltunetable",
        "Description": "Role for use by tunetable to sample tables irrespective of row level security"

      }
```

```
    ]
  }
}
```

You can use the Query method to query any table in Caché. The following call queries the SAMPLES Sample.Person database.

```
POST http://localhost:57772/api/atelier/v1/SAMPLES/action/query
{"query": "SELECT Age,SSN,Home_City,Name FROM Sample.Person WHERE Age = 25"}
```

This call returns:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": [
      {
        "Age": 25,
        "SSN": "230-78-7696",
        "Home_City": "Larchmont",
        "Name": "DeLillo,Jose F."
      },
      {
        "Age": 25,
        "SSN": "546-73-7513",
        "Home_City": "Gansevoort",
        "Name": "Klingman,Thelma H."
      }
    ]
  }
}
```

3

Caché Source Code File REST API Reference

The Caché Source Code File REST interface supports the following methods:

- GetServer: returns information about the server.
- HeadServer: returns the HttpHeader for the server.
- GetJobs: returns a list of running jobs.
- GetMetaData: returns contents of the METADATA.zip file for the named database.
- GetCSPApps: returns a list of CSP applications.
- GetNamespace: returns information about a specific namespace.
- GetDocNames: returns a list of source code file names.
- GetModifiedDocNames: returns a list of source code files that have been modified since the time the database had the specified hash.
- PutDoc: saves the supplied source code file.
- GetDoc: returns the text for the named source code file.
- DeleteDoc: deletes the named source code file.
- HeadDoc: returns the HttpHeader for the named source code file.
- GetDocs: returns the text for the all of the specified source code files.
- DeleteDocs: deletes the list of named source code files.
- Compile: compiles the source code files that you specify.
- Index: returns summary information about the specified source code files.
- Query: performs an SQL query on any Caché table and returns the results.
- Search: searches source code files in the Caché database.
- GetEnsClassType: returns a list of Ensemble class names.
- GetAdpInputOutputClass: returns the input and output type for the adapter.

3.1 GetServer

This method returns information about the server, including Caché Source Code File REST API version and namespaces that are available on the server.

For an example and additional details refer to Getting Information about the Caché Server in the tutorial chapter of this manual.

3.1.1 URL

GET `http://`*server*`:`*port*`/api/atelier/`

3.1.2 JSON Messages

The following returned content is a server descriptor:

```
{
    "status": {
        "errors": [],
        "summary": ""
    },
    "console": [],
    "result": {
        "content": {
            "version": "Cache for Windows (x86-64) 2017.2 (Build 691U) Wed Jun 7 2017 20:45:20 EDT",
            "id": "B70A630D-A34D-43B1-8EA5-EDF8F38992C4",
            "api": 2,
            "features": [
                {
                    "name": "DEEPSEE",
                    "enabled": true
                },
                {
                    "name": "ENSEMBLE",
                    "enabled": true
                },
                {
                    "name": "HEALTHSHARE",
                    "enabled": false
                }
            ],
            "namespaces": [
                "%SYS",
                "DOCBOOK",
                "ENSDEMO",
                "ENSEMBLE",
                "INVENTORY",
                "SAMPLES",
                "USER"
            ]
        }
    }
}
```

3.1.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.2 HeadServer Method

This method returns the HttpHeader for the server.

3.2.1 URL

HEAD `http://`*server*`:`*port*`/api/atelier/`

3.2.2 JSON Messages

No returned content.

3.2.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.3 GetJobs

This method returns a list of running jobs on the Caché instance.

3.3.1 URL

GET `http://`*server*`:`*port*`/api/atelier/v1/%25SYS/jobs`

Note: Because % is a URL special character, to specify a literal % you must follow it with 25 (the hexadecimal code for the percent character). Therefore, you must use %25SYS to specify the literal %SYS.

3.3.2 JSON Messages

The following returned content is an array of job descriptors:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": [
      {
        "pid": 1394,
        "namespace": "%SYS",
        "routine": "%Studio.Debugger.1",
        "state": "RUN",
        "device": "|TCP|1972|1394"
      },
      {
        "pid": 1345,
        "namespace": "%SYS",
        "routine": "RECEIVE",
        "state": "HANG",
        "device": "/dev/null"
      },
      {
```

```
        "pid": 1364,
        "namespace": "%SYS",
        "routine": "%SYS.TaskSuper.1",
        "state": "SELECTW",
        "device": "/dev/null"
      },
      {
        "pid": 1396,
        "namespace": "%SYS",
        "routine": "%SYS.cspServer3",
        "state": "READ",
        "device": "|TCP|1972|1396"
      },
      {
        "pid": 1346,
        "namespace": "%SYS",
        "routine": "ECPWork",
        "state": "RUNW",
        "device": "/dev/null"
      },
      {
        "pid": 1417,
        "namespace": "%SYS",
        "routine": "%SYS.BINDSRV",
        "state": "READ",
        "device": "|TCP|1972|1417"
      }
    ]
  }
}
```


3.3.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.4 GetMetaData

This method returns the binary contents of the METADATA.zip file for the named database. Atelier uses this file to store index information so that it can preserve this information for future sessions.

3.4.1 URL

GET `http://`*server*`:`*port*`/api/atelier/v1/%25SYS/metadata/`*database*

Note: Because % is a URL special character, to specify a literal % you must follow it with 25 (the hexadecimal code for the percent character). Therefore, you must use %25SYS to specify the literal %SYS.

3.4.2 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 404 if the source code file does not exist.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.5 GetCSPApps

This method returns a list of CSP applications defined on the server or defined for a specified namespace on the server.

3.5.1 URL

GET `http://`*server*`:`*port*`/api/atelier/v1/%25SYS/cspapps`

GET `http://`*server*`:`*port*`/api/atelier/v1/%25SYS/cspapps/`*namespace*

Where:

namespace

Specifies the name of the namespace. If *namespace* is not specified, this method returns the CSP applications for all namespaces.

Note: Because % is a URL special character, to specify a literal % you must follow it with 25 (the hexadecimal code for the percent character). Therefore, you must use %25SYS to specify the literal %SYS.

3.5.2 URL Parameters

The URL parameter ?detail=1 can be passed to return an array containing objects which describe the application in more detail.

3.5.3 JSON Messages

The following returned content is an array listing the defined CSP applications:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": [
      "/csp/broker",
      "/csp/sys",
      "/csp/sys/bi",
      "/csp/sys/exp",
      "/csp/sys/mgr",
      "/csp/sys/op",
      "/csp/sys/sec",
      "/isc/studio/rules",
      "/isc/studio/templates",
      "/isc/studio/usertemplates",
      "/csp/docbook",
      "/csp/documatic",
      "/csp/ensdemo",
      "/csp/ensemble",
      "/csp/samples",
      "/csp/user"
    ]
  }
}
```

The following is the same returned content with detail=1:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
```

```
  "result": {
    "content": [
      {
        "name": "/csp/broker",
        "default": false
      },
      {
        "name": "/csp/sys",
        "default": true
      },
      {
        "name": "/csp/sys/bi",
        "default": false
      },
      {
        "name": "/csp/sys/exp",
        "default": false
      },
      {
        "name": "/csp/sys/mgr",
        "default": false
      },
      {
        "name": "/csp/sys/op",
        "default": false
      },
      {
        "name": "/csp/sys/sec",
        "default": false
      },
      {
        "name": "/isc/studio/rules",
        "default": false
      },
      {
        "name": "/isc/studio/templates",
        "default": false
      },
      {
        "name": "/isc/studio/usertemplates",
        "default": false
      },
      {
        "name": "/csp/docbook",
        "default": true
      },
      {
        "name": "/csp/documatic",
        "default": false
      },
      {
        "name": "/csp/ensdemo",
        "default": true
      },
      {
        "name": "/csp/ensemble",
        "default": true
      },
      {
        "name": "/csp/samples",
        "default": true
      },
      {
        "name": "/csp/user",
        "default": true
      }
    ]
  }
}
```


3.5.4 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.6 GetNamespace

This method returns information about a specific namespace.

3.6.1 URL

GET `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*

3.6.2 JSON Messages

The following is the returned content information about the namespace DOCBOOK:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": {
      "name": "DOCBOOK",
      "db": [
        { "name": "DOCBOOK", "crhash": "5046B9BF0DE", "default": true },
        { "name": "CACHESYS", "crhash": "47763751EC", "default": false },
        { "name": "CACHE", "crhash": "4776EDD1C3", "default": false },
        { "name": "CACHELIB", "crhash": "5023332D0A7", "default": false }
      ],
      "features": [
        {
          "name": "ENSEMBLE",
          "enabled": false
        }
      ],
    }
  }
}
```

3.6.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.7 GetDocNames

This method returns a list of source code file names. The optional *cat* and *type* constrain the types of source code files.

For an example and additional details refer to Getting the Source Code Files Defined in a Namespace in the tutorial chapter of this manual.

3.7.1 URL

GET `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/docnames`

GET `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/docnames/`*cat*

GET `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/docnames/`*cat*`/`*type*

Where:

cat

Specifies a category code: CLS = class; RTN = routine; CSP = csp file; OTH = other. Default is *.

type

Specifies a source code file type. Can be an * wildcard or a three-letter code. For CLS, type must be *. For RTN, type may be mac, int, inc, bas ,mvi, or mvb. For CSP, type can be a list of file types such as js or css separated by commas. Default is *.

3.7.2 URL Parameters

- The URL parameter 'generated=1' specifies that generated source code files should be included.
- The URL parameter 'filter' provides a SQL filter that can be used to match the names.

3.7.3 JSON Messages

The following is the returned content, an array of source code file descriptors:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": [
      {
        "name": "%Activate.Enum.cls",
        "cat": "CLS",
        "ts": "2016-08-03 20:01:42.000",
        "upd": true,
        "db": "CACHELIB",
        "gen": false
      },
      ...
      {
        "name": "EnsProfile.mac",
        "cat": "RTN",
        "ts": "2003-09-19 13:53:31.000",
        "upd": true,
        "db": "INVENTORYR",
        "gen": false
      },
      ...
      {
        "name": "xyz.mac",
        "cat": "RTN",
        "ts": "2016-08-11 15:05:02.167",
        "upd": false,
        "db": "INVENTORYR",
        "gen": false
      }
    ]
  }
}
```

3.7.4 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.8 GetModifiedDocNames

This method returns a list of source code files that have been modified since the time the databases had the specified hashes. It is passed a list of database keys and hashes as a JSON array. The hash values are used to determine if anything has changed in the database defined by the key. Typically, you first call this api with an empty array as the incoming JSON message. This returns the names of all source code files in the namespace with the database key and hash for each file. Then you can post the dbname and dbhash to discover which source code files have been modified on the server since the last call.

You post the list of source code files to check as shown in the following example:

```
[ { "dbname" : "USER",
    "dbhash" : "KWAGbOdnRblPzANaiv1Oiu0BZLI"
}, ... ]
```

3.8.1 URL

POST `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/modified/`*type*

Where:

type

Specifies a source code file type as * or a three-letter code, ls, mac, int, inc, bas, or mvi. Default is *.

This call requires the header `Content-Type application/json`.

3.8.2 JSON Messages

The following is the returned content, an array of source code file descriptors:

```
[ { "dbname" : "USER",
    "dbhash" : "Qx1zuNaulq3b_1yR9ahZAfjkc-",
    "crhash" : "47763751EC",
    "docs": [{
            "name": "User.NewClass1.cls",
            "ts": "2016-01-04 14:00:04.000",
            "gen": false,
            "depl": false
            }, ... ]
   }, ... ]
```

If a source code file has been deleted since the specified dbhash, it is returned in the list with a time stamp set to an empty string:

```
"ts": ""
```

If a database was included because of mapping and the mapping is removed, both dbhash and crhash are returned with a "000" value and docs is returned as an empty array.

3.8.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 400 if the posted content is empty or *type* is anything other than CLS.
- HTTP 415 if content type is not application/json.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.9 PutDoc

This method saves the supplied source code file. If the file does not exist, this method creates it, and, if the file exists, this method replaces the existing file with the one specified. To ensure that you are overwriting the correct version of the file, specify the If-None-Match header with the timestamp value returned in the ETAG header of a previous PutDoc or GetDoc. If you want to overwrite the file without checking the version, specify the ?ignoreConflict=1 URL parameter. This method returns a corresponding source code file object. If you are saving a binary file, set the `enc` element of the incoming JSON message to true and include the file content as an array of blocks of base64. If the text on the server is changed during the save process (for example by a source control hook) the updated text will be returned in the content array of the returned source code file.

Errors pertaining to the source code file will be in the status property of the returned source code file object.

Version 2 PutDoc has the capability to accept the contents of the file in three formats: the default UDL format, XML format, and the format used by the legacy %RO export utility. PutDoc automatically recognizes the format of the file contents.

For an example and additional details refer to Creating a New File in a Namespace or Updating an Existing File in the tutorial chapter of this manual.

3.9.1 URL and Input JSON Message

PUT `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/doc/`*doc-name*

PUT `http://`*server*`:`*port*`/api/atelier/v2/`*namespace*`/doc/`*doc-name*

The following is an example of the input JSON message for a PutDoc for the source code file xyz.mac:

```
{
 "enc": false,
 "content": [
   "ROUTINE xyz",
   "xyz ;",
   "   w \"hello\""
   ]
}
```

Note: If you are creating a CSP file, the value of *doc-name* includes the / (slash) character. This is the reason that the URLMap defining PutDoc contains a (.*) for this parameter name instead of :docname. See "Creating the URL Map for REST" in *Creating REST Services in Caché* for details.

3.9.2 URL Parameters

The URL parameter ?ignoreConflict=1 can be passed to bypass ETAG checking. This forces the source code file to be written to the server even if the file has changed since you previously accessed it.

3.9.3 HTTP Headers

- If-None-Match—To ensure that you are overwriting the correct version of the file, specify the If-None-Match header with the timestamp value returned in the ETAG header of a previous PutDoc or GetDoc.

3.9.4 JSON Messages

The following is the returned content for a PUT of source code file xyz.mac:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "name": "xyz.mac",
    "db": "INVENTORYR",
    "ts": "2016-09-14 14:10:16.540",
    "upd": false,
    "cat": "RTN",
    "status": "",
    "enc": false,
    "flags": 0,
    "content": []
  }
}
```


When a class is saved PutDoc always returns the storage section because it may be normalized by the server. The 'flags' json field will be 1 if the contents is only the storage section. 'flags' will be 0 if PutDoc returns either the entire class in content or if content is empty.

3.9.5 HTTP Return Codes

- HTTP 200 if updated.
- HTTP 201 if created.
- HTTP 400 if the resource name is an invalid source code file name.
- HTTP 404 if the resource is not found.
- HTTP 409 if a conflict between server and client versions is detected based on the timestamp. If the file exists, PutDoc returns this code unless the If-None-Match header specifies the current timestamp value of the file on the server. If the conflict exists, the return message includes the contents of the source code file on the server.
- HTTP 415 if not passed text/plain as content type.
- HTTP 425 if the source code file is locked and cannot be written to.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.10 GetDoc

This method returns the text for the named source code file and namespace.

Return content will contain a source code file object.

Errors pertaining to the source code file will be in the status property of the source code file object. If source control hooks are enabled for the namespace any console output generated by the hook will be captured and returned as an array of lines in the 'console' array.

The result contains the name of the requested file, the database where it is stored, its timestamp, and its category abbreviation (CLS = class; RTN = routine; CSP = csp file; OTH = other), as well as the source code file contents which are returned in an array.

- For text files this will be an array of strings and the 'enc' json field will be set to false.
- For binary files this will be an array of base64 encoded chunks and the 'enc' field will be set to true.

Version 2 GetDoc can return the file contents in UDL format (the default), XML format, or the format used by the legacy %RO export utility.

For an example and additional details refer to Getting the Source Code Files Defined in a Namespace in the tutorial chapter of this manual.

3.10.1 URL

GET `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/doc/`*doc-name*

GET `http://`*server*`:`*port*`/api/atelier/v2/`*namespace*`/doc/`*doc-name*

Note: If you are getting a CSP file, the value of *doc-name* includes the / (slash) character. This is the reason that the URLMap defining GetDoc contains a (.*) for this parameter name instead of :docname. See "Creating the URL Map for REST" in *Creating REST Services in Caché* for details.

3.10.2 URL Parameters

- The URL parameter ?binary=1 can be passed to force the source code file to be encoded as binary.
- The URL parameter ?storageOnly=1 can be passed to return only the storage portion of a class.
- In version 2, the URL parameter ?format= parameter can be passed to specify that the contents of the file should be returned in UDL format (the default), XML format, or the format used by the legacy %RO export utility.
  - ?format=udl
  - ?format=xml
  - ?format=%RO

  If you specify ?binary=1, GetDoc ignores the format parameter.

3.10.3 HTTP Headers

- If-None-Match—Specify the value returned in the HTTP ETAG header from the previous call to GetDoc or PutDoc for this file. If the file has not changed since the previous call, GetDoc returns the HTTP 304 status.

3.10.4 JSON Messages

The following is an example of the result returned by requesting %Activate.Enum.cls:


```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "name": "%Activate.Enum.cls",
    "db": "CACHELIB",
    "ts": "2016-09-13 22:31:24.000",
    "upd": true,
    "cat": "CLS",
    "status": "",
    "enc": false,
    "flags": 0,
    "content": [
      "/// This class is the superclass for all enumerated types generated from",
      "/// a type library",
      "Class %Activate.Enum Extends %Integer [ Not ProcedureBlock, System = 3 ]",
      "{",
      "",
      "}",
      ""
```

```
    ]
  }
}
```

The following is the result of the same request with ?binary=1:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "name": "%Activate.Enum.cls",
    "db": "CACHELIB",
    "ts": "2016-01-04 14:00:04.000",
    "cat": "CLS",
    "status": "",
    "enc": true,
    "content": [
      "Ly8vIFRoaXMgY2xhc3MgaXMgdGhlIHN1cGVyY2xhc3MgZm9yIGFsbCBl  ... PSAzIF0KewoKfQo="
    ]
  }
}
```

3.10.5 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 304 if the source code file has not been modified (see https://en.wikipedia.org/wiki/HTTP_ETag).
- HTTP 400 if the named resource is not a valid source code file name.
- HTTP 404 if the source code file does not exist.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

If a 'soft' error occurs such as a 'source code file does not exist', additional information can be found in the 'status' field of the result. Examples of other soft errors include 'file is locked'. For example, the following could be returned with an HTTP 404 return code:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "name": "xyz1.mac",
    "db": "",
    "ts": "",
    "cat": "RTN",
    "enc": false,
    "content": "",
    "status": "ERROR #16005: Document 'xyz1.mac' does NOT exist"
  }
}
```

3.11 DeleteDoc

This method deletes the named source code file in the specified namespace. It returns the corresponding source code file object.

Errors pertaining to the source code file will be in the status property of the source code file object.

For an example and additional details refer to Deleting a File in the tutorial chapter of this manual.

3.11.1 URL

DELETE `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/doc/`*doc-name*

Note: If you are deleting a CSP file, the value of *doc-name* includes the / (slash) character. This is the reason that the URLMap defining DeleteDoc contains a (.*) for this parameter name instead of :docname. See "Creating the URL Map for REST" in *Creating REST Services in Caché* for details.

3.11.2 JSON Messages

The following is the returned content for a DELETE of source code file xyz.mac:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "name": "xyz.mac",
    "db": "INVENTORYR",
    "ts": "",
    "cat": "RTN",
    "status": "",
    "enc": false,
    "flags": 0,
    "content": []
  }
}
```

3.11.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 400 if the named resource is not a valid source code file name.
- HTTP 404 if the source code file does not exist.
- HTTP 423 if the resource is locked.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.12 HeadDoc

This method returns the HttpHeader for the named source code file and namespace. This header contains a timestamp which can be used to detect discrepancies between server and client versions.

3.12.1 URL

HEAD `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/doc/`*doc-name*

Note: If you are getting the HTTP header for a CSP file, the value of *doc-name* includes the / (slash) character. This is the reason that the URLMap defining HeadDoc contains a (.*) for this parameter name instead of :docname. See "Creating the URL Map for REST" in *Creating REST Services in Caché* for details

3.12.2 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 400 if the resource name is an invalid source code file name.
- HTTP 404 if the resource is not found.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.13 GetDocs

This method returns the text for the all of the specified source code files in the namespace.

3.13.1 URL

POST `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/docs`

A list of source code files to be fetched is passed in the body of the http request. The request body is a JSON array of names of source code files you want to fetch. For example, `[ "%Activate.Enum.cls", ... ]`.

This call requires the header `Content-Type application/json`.

3.13.2 JSON Messages

Return content is an array of source code file objects. See GetDoc method for an example of the structure of a source code file object.

Errors pertaining to a source code file will be in the status property of each source code file object. This method does NOT support the storageOnly flag. Neither does it do ETAG checking (and therefore will not return an HTTP 304 under any circumstances).

3.13.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 415 if the passed content type is not application/json.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.14 DeleteDocs

This method deletes a list of named source code files. It returns a corresponding array of source code file objects.

3.14.1 URL

DELETE `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/docs`

The list of files to delete is passed in the body of the http request as a JSON array. For example, `[ "%Activate.Enum.cls", ... ]`.

This call requires the header `Content-Type application/json`.

3.14.2 JSON Messages

The following is the returned content for a DELETE of source code file xyz.mac and the nonexistent class notexist.cls:

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [
  ],
  "result": [
    {
      "name": "xyz.mac",
      "db": "INVENTORYR",
      "status": ""
    },
    {
      "name": "notexist.cls",
      "db": "",
      "status": "ERROR #5001: Document Does Not Exist: User.notexist.cls
[zExistsDoc+3^%Atelier.v1.Utils.General.1:%SYS]"
    }
  ]
}
```

Errors pertaining to a each source code file will be in the status property of each returned source code file object. If the status is an empty string the source code file was deleted successfully. Otherwise the source code file was NOT deleted.

For deleted source code files, the db property will indicate from which database the doc was deleted.

3.14.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 400 if the posted data does not contain a JSON array.
- HTTP 415 if the passed content type is not application/json.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.15 Compile

This method compiles source code files. It permits the compilation of more than one source code file at a time. It returns an array of corresponding source code file objects.

The list of files to be compiled is passed in the body of the http request as a JSON array. For example, `[ "%Activate.Enum.cls", ... ]`.

For an example and additional details refer to Compiling a File in the tutorial chapter of this manual.

3.15.1 URL

POST `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/action/compile`

This call requires the header `Content-Type application/json`.

3.15.2 URL Parameters

- The URL parameter 'flags' can be passed (default "cuk") which will be passed to the compiler.
- The URL parameter 'source' can be passed with a value of 0 if you don't want the source of the compiled source code file to be returned.

3.15.3 JSON Messages

The following is the returned content when compiling Atelier.NewClass1:


```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [
    "Compilation started on 01/12/2016 17:44:00 with qualifiers 'cuk'",
    "Compiling class Atelier.NewClass1",
    "Compiling table Atelier.NewClass1",
    "Compiling routine Atelier.NewClass1.1",
    "Compilation finished successfully in 0.067s.",
    ""
  ],
  "result": {
    "content": [
      {
        "name": "Atelier.NewClass1.cls",
        "status": "",
        "content": [
          "Storage Default",
          "{",
          "<Data name=\"NewClass1DefaultData\">",
          "<Value name=\"1\">",
          "<Value>%%CLASSNAME</Value>",
          "</Value>",
          "</Data>",
          "<DataLocation>^Atelier.NewClass1D</DataLocation>",
          "<DefaultData>NewClass1DefaultData</DefaultData>",
          "<IdLocation>^Atelier.NewClass1D</IdLocation>",
          "<IndexLocation>^Atelier.NewClass1I</IndexLocation>",
          "<StreamLocation>^Atelier.NewClass1S</StreamLocation>",
          "<Type>%Library.CacheStorage</Type>",
          "}",
          ""
        ],
        "db": "CACHESYS",
        "ts": "2016-01-12 17:44:00.053",
        "enc": false,
        "flags": 1
      }
    ]
  }
}
```


Errors pertaining to a source code file will be in the status property of each source code file object.

If compiling a persistent class causes the storage definition to change, the storage definition is returned as the content of a source code file object. Otherwise the result content will be empty.

3.15.4 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 400 if the resource name is an invalid source code file name.
- HTTP 404 if the resource is not found.
- HTTP 423 if the source code file is locked.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.16 Index

This method returns summary information on the specified source code files. Your application can use this information to create an index to the source code files. It returns an array of index source code file objects.

The list of source code files to be indexed is passed in the body of the http request. The request body is a JSON array of names of source code files. For example, `[ "%Activate.Enum.cls", ... ]`.

3.16.1 URL

POST `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/action/index`

This call requires the header `Content-Type application/json`.

3.16.2 JSON Messages

Errors pertaining to a source code file are in the status property of each source code file object. The returned array contains information relating to the structure and documentation of source code files on the server. It will vary by the category to which the source code file belongs. The following is an example for a class (category CLS). (Currently we only support the indexing of classes.):

```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": [
      {
        "name": "%Activate.GenericObject.cls",
        "db": "CACHELIB",
        "ts": "2016-01-04 14:00:04.000",
        "gen": false,
        "others": [
          "%Activate.GenericObject.1.INT"
        ],
        "cat": "CLS",
        "content": {
          "desc": "This class provides functionality to create an ActiveX object, invoke its methods
and Get/Set its properties by name.",
          "depl": false,
          "depr": false,
          "final": false,
          "hidden": false,
          "super": [
            "%Activate.IDispatch"
          ],
          "methods": [
            {
              "name": "CreateObject",
              "desc": "This method is used to create a generic object given only its progid. If the
object cannot be found an exception is thrown.
                       The return value should be tested against $$$NULLOREF  in the usual manner to
ensure that the object has been successfully created",
              "depr": false,
              "final": true,
              "internal": false,
              "private": false,
              "scope": "class",
              "returntype": "%Library.RegisteredObject",
              "args": [
                {
                  "name": "Progid",
                  "type": "%Library.String"
                }
              ]
            },
            {
              "name": "GetObject",
```

```
              "desc": "This method is used to create a generic object from a moniker. If the object
cannot be found an exception is thrown.
                       The return value should be tested against $$$NULLOREF  in the usual manner to
ensure that the object has been successfully created.",
              "depr": false,
              "final": true,
              "internal": false,
              "private": false,
              "scope": "class",
              "returntype": "%Library.RegisteredObject",
              "args": [
                {
                  "name": "Moniker",
                  "type": "%Library.String"
                }
              ]
            }
          ],
          "parameters": [],
          "properties": []
        },
        "status": ""
      }
    ]
  }
}
```


3.16.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 415 if the passed content type is not application/json.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.17 Query

This method performs an SQL query on a Caché table and returns the results. The request body is a JSON object which specifies the query. It returns an array of objects that match the query conditions. Each returned object contains information relating to one row returned by the query.

3.17.1 URL

POST `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/action/query`

The SQL query is specified in the body of the URL request. The query must specify a database in the specified namespace.

This call requires the header `Content-Type application/json`.

3.17.2 JSON Messages

Return content is an array of objects. Errors will be in the status property of each source code file object:


```
{
  "status": {
    "errors": [],
    "summary": ""
  },
  "console": [],
  "result": {
    "content": [
      {
        "ID": "%all",
        "Description": "The Super-User Role"
      },
      {
```

```
        "ID": "%db_%default",
        "Description": "R/W access for this resource"
      },
      {
        "ID": "%db_cache",
        "Description": "R/W access for this resource"
      },
      ...
      {
        "ID": "%sqltunetable",
        "Description": "Role for use by tunetable to sample tables irrespective of row level security"

      }
    ]
  }
}
```

3.17.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 415 if the passed content type is not application/json.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.18 Search

This method finds files in the Caché database with the specified contents. The Search method is available in version 2 of the API. This method returns the results of the search in a format intended to be displayed to the user.

3.18.1 URL

POST `http://`*server*`:`*port*`/api/atelier/v2/`*namespace*`/action/search`

3.18.2 URL Parameters

- The required URL parameter ?query=*expression* specifies a regular expression or a text string to search for in the specified files.
- The required URL parameter ?files=*file-list* provides a comma-separated list of files or file masks, such as al*.mac, to search for the specified expression.
- The optional URL parameter ?regex=1 specifies that the query URL parameter contains a regular expression and is the default. ?regexp=0 specifies that the query contains a text string and should not be interpreted as a regular expression.
- The optional URL parameter ?sys=1 specifies to include system files in the search. The default is ?sys=0, which excludes system files.
- The optional URL parameter ?gen=1 specifies to include generated files in the search. The default is ?gen=0, which excludes generated files.
- The optional URL parameter ?max=*integer* specifies the maximum number of results to return. The default is ?max=200.

3.18.3 JSON Messages

The following search REST call searches all .cls and .mac files for the word “Email” preceded and followed by a space. (In a regular expression, \s is matched by a space character.)

```
GET
localhost:57772/api/atelier/v2/SAMPLES/action/search?query=.*\sEmail\s.*&files=*.cls,*.mac
```

This call returns the following message. The return message may vary based on the contents of the SAMPLES namespace.

```
{
    "status": {
        "errors": [],
        "summary": ""
    },
    "console": [
        "",
        "Searching for '.*\\sEmail\\s.*' in '*.cls,*.mac'",
        "Wasabi.Data.Employee.cls(Email): Property Email ",
        "Wasabi.Person.API.Employee.cls(Email): Property Email ",
        "ZAUTHENTICATE.mac(175): Properties(\"EmailAddress\") - Email address",
        "Found 3 occurrence/s in 3 file/s."
    ],
    "result": [
        {
            "doc": "Wasabi.Data.Employee.cls",
            "matches": [
                {
                    "member": "Email",
                    "text": "Property Email "
                }
            ]
        },
        {
            "doc": "Wasabi.Person.API.Employee.cls",
            "matches": [
                {
                    "member": "Email",
                    "text": "Property Email "
                }
            ]
        },
        {
            "doc": "ZAUTHENTICATE.mac",
            "matches": [
                {
                    "line": "175",
                    "text": "Properties(\"EmailAddress\") - Email address"
                }
            ]
        }
    ]
}
```

3.18.4 HTTP Return Codes

- HTTP 200 if the request is valid.
- HTTP 400 if there are missing required URL parameters.

3.19 GetEnsClassType

This method returns a list of Ensemble class names.

3.19.1 URL

GET `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/ens/classes/`*type*

Where:

type

is an integer and returns classes corresponding to that integer as follows:

Adapters 1

InboundAdapters 2

OutboundAdapters 3

Messages 4

Requests 5

Responses 6

BusinessServices 7

BusinessProcesses 8

BusinessOperations 9

DataTransformation 10

Production 11

BusinessHost 12

Dashboard 13

Rule 14

3.19.2 JSON Messages

The following returned content is an array of Ensemble class names:

```
{
  status: {
    errors: []
    summary: ""
  }
  console: []
  result: {
    content: [
        "Ens.Enterprise.MsgBank.BankTCPAdapter",
        "Ens.Enterprise.MsgBank.ClientTCPAdapter",
        "Ens.InboundAdapter",
        "Ens.OutboundAdapter"
    ]
  }
}
```

3.19.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 500 if an unexpected error occurred (details will be in status error array).

3.20 GetAdpInputOutputClass

This method returns the input and output type for a specified Ensemble adapter.

3.20.1 URL

GET `http://`*server*`:`*port*`/api/atelier/v1/`*namespace*`/ens/adapter/`*name*

3.20.2 JSON Messages

The following is an example of returned content:

```
{
  status: {
    errors: []
    summary: ""
  }
  console: []
  result: {
    content: {
      input: "%Stream.Object"
      output: "%String"
    }
  }
}
```

3.20.3 HTTP Return Codes

- HTTP 200 if OK.
- HTTP 404 if the adapter does not exist.
- HTTP 500 if an unexpected error occurred (details will be in status error array).