

SAFIS Unified API

Overview 2

USER_PERMITS - retrieve a list of permits for the user 6

TRIPS_UPLOAD - send a set of trips rows to safis 8

TRIPS_UPLOAD_ERRORS - fetch a list of errors from trip rows that did not load 11

NEG_TRIPS_UPLOAD - send a set of negative trip reports to safis 13

NEG_TRIPS_UPLOAD_ERRORS - fetch a list of errors from negative reports that didn't load 15

LOOKUP_LIST - fetch a set of rows for a specified lookup table 16

LOOKUP_LIST_DATE_CHANGED - fetch the last date a specified lookup table was changed 20

PERMIT_COMPLIANCE - checks the current compliance status of a dealer permit 22

REPORTS_UPLOAD - send a set of rows to safis 23

REPORTS_UPLOAD_ERRORS - fetch a list of errors from rows that did not load 26

PERMIT_ENDORSEMENTS - fetch a list of endorsements for a given a partner 27

NEG_REPORTS_UPLOAD - send a set of negative dealer reports to safis 29

NEG_REPORTS_UPLOAD_ERRORS - fetch a list of errors from negative dealer reports that didn't load 31

PARTICIPANTS - check for users with the given information 32

Overview

This API is used by remote applications to communicate with the SAFIS backend database server. It is not intended for any other purpose, for example it does not supply a web-page interface.

This document is intended as a programmer's reference manual. It is used by both application developers who build a front-end, and the Oracle PL/SQL programmers who maintain this API back-end. Using it assumes you are already familiar with the upload data but do not need any understanding of underlying Oracle technology.

Data is passed in in JSON format, and returned the same way.

A remote application using this API can be either web based, or a mobile application in an embedded device or a phone, so we choose to use the term "device" to refer to the remote.

COMMAND MECHANISM

This API breaks the interaction between device and server into "commands". The device sends a command to the server and gets back a status message, and possibly some data in a pre-specified format. This response is formatted as a JSON object. The API uses a common format for this described in detail later.

All of these commands are detailed, with examples, later in this document.

The interaction mechanism for all commands is the same, as follows:

- The mobile app submits a page to a URL
- The domain-name, TCP-port-number and base-URL are all pre-defined. The connection between web and oracle is supplied by an oracle restful service.
- The specific command is added to the base URL (see examples below)
- The "payload" of the request is passed as a the body of the http POST request using the JSON object format.
- The database will return the results formatted as a JSON object.
- Each specific command will have its own result-object-format (see below)
- The result will always contain a STATUS
- No data found will result in returning an empty array, i.e. no data rows but there will be a "row" object. So the returned row-array length will be zero.
- All errors will return a STATUS format, this format will be shared by all commands. It will include an error code and a text description.
- The result is a JSON object. Note that this is NOT a valid piece of html, there is no markup, so it will not display correctly in a browser window.

BASE URL

For test and production these will be:

- protocol: https
- domain & base URL:
 - test - "safis.accsp.org:8443/safis_test/safist/api/"
 - prod - "safis.accsp.org:8443/safis_prod/safis_prod/api/"
- Command: The command is added to the base-url separated with a slash. All commands are implemented as a procedure in a single package called "API".

- Parameters: for an http POST operation these are all embedded in the body of the http request as a JSON object.

REQUIRED PARAMETERS

All commands require basic HTTP authentication. All commands requesting or receiving confidential data will supply a an API key along with the username and password of the end-user. Access to non-user specific data, like the gears for a given partner, only requires an API. The format for HTTP Basic Authentication is as follows.

Confidential-user-specific data

Username: my_user_name@my_api_key

Password: my_password

Non-confidential non-user specific data

Username: @my_api_key

Password: <<Leave it blank>>

The use of basic authentication makes the API truly stateless and therefore each transaction request is self-contained and there are no sessions to track. Also there are no user login and user logout function as these are unnecessary.

ALL parameters are sent to the server in their prescribed JSON format. The order does not matter.

JavaScript and JSON

The API is designed to be callable from a JavaScript application, usually using the jQuery library.

All responses are a single JSON object, this encapsulates a status code and description and a set of data rows.

Example JSON call

To make the call from a JavaScript function using JQuery a typical call would look like this:

JavaScript

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
```

```
<script>
```

```
function callSomething(){
```

```
var mydata = JSON.stringify({"p_usertype": "FISHER"});
```

```
var my_user = "username@myapikey";
```

```
var my_pass = "password";
```

```
$.ajax (
```

```
{ url: "https://safis.accsp.org:8443/safis_test/safist/user_permits",
```

```
  data: mydata,
```

```
  async: true,
```

```
  dataType: "json",
```

```
  contentType: "application/json; charset=utf-8",
```

```
  type: "POST",
```

```
  headers: {'Authorization': 'Basic ' + window.btoa(my_user + ':' + my_pass)}
```

```
},
```

```
  success: function(data, textStatus, jqXHR) {
```

```
    console.log("you got in");
```

```
  },
```

```
  error: function(jqXHR, textStatus, errorThrown) {
```

```
    console.log("you got 99 problems:" + JSON.stringify(jqXHR));
```

```
  }
```

```
};
```

```
);
```

```
}
```

```
</script>
```

```
</script>
```

Example JSON response

A typical response from this call would be:

```
"row": [  
  {  
    "license_nbr": "123",  
    "license_type": "FL_COMMERCIAL",  
    "permit_type": "Commercial",  
    "agency": "0015",  
    "in_use": "Y"  
  }  
],  
"OK": "User :flfisher Active Permit Count:1"  
}
```

In this example the response is read in as a JavaScript object called “data” as an argument to a function called “callback_success()” that must exist in the same javascript page.

The JavaScript developer does not really need to know anything about the internal workings, all they need is to know what parameters to pass the API, and what to expect back. All API calls always return a status and description, if there is no data they return a “row” object that is empty, so the .rows.length property will be zero.

Remember to replace the username, password and API key in the example above with valid values.

The remainder of this document is a description of each command, in a standard format. Each has examples and sample output. The description of each contains notes on when and how it is intended to be used.

USER_PERMITS - retrieve a list of permits for the user

This call retrieves a list of dealer permits for the logged in user with a valid username, password and API key.

Parameters:

P_USERTYPE char, required, no wildcards.

Description

The user type is passed in with the P_USERTYPE parameter. The type of PERMIT being requested is sent in as P_USERTYPE. The only current valid values of P_USERTYPE are *DEALER* and *FISHER*. As the API is expanded to accommodate more types of users, the list will grow.

Authorization Header:

username@apikey:password

Example Request Body:

```
{
  "p_usertype":"DEALER"
}
```

Returns

```
row[n].DATA
  .status
  .description
```

Row Response

It returns a list of license numbers and corresponding agency for the user whose user type is passed in. A status of OK is returned, with a single row that has the username and the total count of licenses being returned.

.row[n].	
license_nbr	the license number is the SAFIS database for a given permit
License_type	The license type assigned by the partner
permit_type	The type of permit being returned. There are 4 current possible values: <i>Dealer</i> (for all dealer permits) <i>Commercial</i> (Commercial Fisher permits) <i>For-Hire</i> (For-Hire Fisher permits) <i>Operator</i> (Operator permits for federally permitted fishermen)
agency	The issuing agency of the corresponding license number above.

<code>in_use</code>	A flag to indicate whether the permit is active or not: 'Y' or 'N' values.
<code>.status</code>	
<code>.description</code>	

TRIPS_UPLOAD - send a set of trips rows to safis**Parameters:**

TRIPS table of "trips" records (see description below)

Description:

The calling app will compose a table of trip records to upload, note that since they are submitted as the body of an http POST request they are all char data even if the underlying datatype is date or number.

The data values are submitted as a JSON array of trip objects each containing a nested array of 1 or more efforts and each effort optionally containing an array of 0 or more catch objects.

Below is a list of the valid columns expected to make a complete row. are:

(R) denotes required field for all partners.

Trip level information

CF_LICENSE_NBR (R)	NUM_ANGERS
ISS_AGENCY (R)	VTR_NUMBER
TRIP_TYPE (R)	VESSEL_PERMIT
SUPPLIER_TRIP_ID (R)	SUB_TRIP_TYPE
PORT (R)	REPORTING_SOURCE
STATE (R)	FUEL_USED
COAST_GUARD_NBR (R)	FUEL_PRICE
STATE_REG_NBR (R)	CHARTER_FEE
TRIP_START_DATE (R)	TRIP_ACTIVITY
TRIP_START_TIME (R)	PARTICIPANT_ID
TRIP_END_DATE	
TRIP_END_TIME	
NUM_CREW	

Effort level information

DISTANCE (R)	HOURS_DAYS (R)
IN_STATE (R)	TOTAL_GEAR
AREA_CODE (R)	GEAR_SIZE
SUB_AREA_CODE (R)	MESH_RING_LENGTH
LOCAL_AREA_CODE	MESH_RING_WIDTH
LATITUDE	STRETCH_SIZE
LONGITUDE	TARGET_SPECIES
GEAR (R)	AVG_DEPTH
LMA	SUPPLIER_EFFORT_ID (R)
GEAR_QUANTITY	GEAR_ATTRIBUTES[]
GEAR_SETS	
FISHING_HOURS (R)	

Gear Attributes level information

CODE
VALUE
UNITS

Catch level information

SPECIES_ITIS (R)	DATE_SOLD	SUPPLIER_CATCH_ID (R)
DISPOSITION (R)	REPORTED_QUANTITY (R)	NBR_OF_FISH
MARKET_CODE	PRICE	LENGTH
GRADE_CODE	DEA_ISS_AGENCY	UNIT_OF_LENGTH
UNIT_OF_MEASURE (R)	CATCH_SOURCE	OFFLOAD_PORT
SALE_DISPOSITION_FLAG	CATCH_LATITUDE	
DEALER_LICENSE_NBR	CATCH_LONGITUDE	

This API returns an event_id to the calling application, which can be used to check the status of the upload, and once it is finished The EVENT ID can be used to fetch any records that had errors.

Authorization Header:

username@apikey:password

Example Request Body:

```
{
  "trips": [
    {
      "cf_license_nbr": "644452120",
      "iss_agency": "0013",
      "trip_type": "P",
      "supplier_trip_id": "1",
      "port": "34045",
      "state": "23",
      "coast_guard_nbr": "GX83838",
      "state_reg_nbr": "SC400123",
      "trip_start_date": "07/19/2018",
      "trip_start_time": "0000",
      "trip_end_date": "07/19/2018",
      "trip_end_time": "0000",
      "num_crew": null,
      "num_anglers": "4",
      "vtr_number": null,
      "efforts": [
        {
```

```

"distance": "Estuarine",
"gear": "310",
"gear_quantity": "q",
"gear_sets": null,
"fishing_hours": "5",
"hours_days": "H",
"target_species": "163997",
"avg_depth": "14.583362500",
"attributes": [{"code": "231", "value": "1233", "units", "CM"},
               {"code": "040", "value": "Y"}],

```

```

"catches": [
  {
    "species_itis": "170425",
    "disposition": "001",
    "unit_of_measure": "LB",
    "market_code": "UN",
    "grade_code": "01",
    "reported_quantity": "25",
    "nbr_of_fish": "4",
    "supplier_catch_id": "6"
  },
  {
    "species_itis": "170425",
    "disposition": "404",
    "unit_of_measure": "CN",
    "market_code": "UN",
    "grade_code": "01",
    "reported_quantity": "3",
    "nbr_of_fish": null,
    "supplier_catch_id": "6"
  }
]
}
]
}
]
}

```

Row response

a status of OK is returned, with a single row that has same format at the UPLOAD_LIST API, but for just this one upload

.row[n].	
event_id	the unique job-id /event_id for this upload assigned by the system
upload_date	when the upload was received
process_date	When the upload was processed. This might be some time after the user submitted it
total_count	count of the trip/effort/catch combinations for this event_id
errored_count	count of rows that had an error of some kind. If the value is more than 0, TRIP_UPLOAD_ERRORS should be called to get the details
trips_accepted[m]	An array objects of successfully entered trips with indicating the assigned trip ids.
supplier_trip_id	
accsp_trip_id	
.status	
.description	

Example Response Body:

```
{
  "row": [
    {
      "event_id": "121582",
      "upload_date": "2018-08-30 12:56:28",
      "process_date": "12:56:32",
      "total_count": "2",
      "errored_count": "0",
      "trips_accepted": [ {"supplier_trip_id": "0192929", "accsp_trip_id": 1231213},
                          {"supplier_trip_id": "0141229", "accsp_trip_id": 1211213} ]
    }
  ],
  "status": "OK",
  "description": "UPDATE EVENTS UPLOAD"
}
```

TRIPS_UPLOAD_ERRORS - fetch a list of errors from trip rows that did not load

Parameters:

P_EVENT_ID id from previous upload_put or upload_list command

Description:

Fetch any errors in the records from this event id. Returned as a JSON encoded object. If no rows had errors there will be no data, and status record will indicate zero errors

Authorization Header:

username@apikey:password

Example Request Body:

```
{  
  "p_event_id":"54321"  
}
```

Rows response

If there are no errors a status response of OK is returned, with a no rows. In this case the .rows.length property is zero.

The format returned is has 3 rows with addition of an ERR_DESC field at the start. The first row is used to hold the explanation of what is wrong with the record. The 2nd row can either be the supplier trip, effort or catch id whichever is appropriate and helps reference the original row passed by trips_upload. Event_id is the event id number given by trips_upload identifying the upload.

```
row[n].ERR_DESC  
  SUPPLIER_CATCH_ID (if applicable)  
  SUPPLIER_EFFORT_ID (if applicable)  
  SUPPLIER_TRIP_ID (if applicable)  
  EVENT_ID  
  
.status  
.description
```

NEG_TRIPS_UPLOAD - send a set of negative trip reports to safis

Parameters:

NEG_TRIPS table of neg_trips records (see description below)

Description:

The calling app will compose a table of negative trip records to upload, note that since they are submitted as the body of an http POST request they are all char data even if the underlying datatype is date or number.

The data values are submitted as a collection of rows with each value denoting the type and column number it belongs.

Below is a list of the valid columns expected to make a complete row. All fields but the end date are required:

CF_LICENSE_NBR - (valid alphanumeric field)
 ISS_AGENCY - (valid alphanumeric field)
 START_DATE - (MM/DD/YYYY)
 END_DATE - (MM/DD/YYYY) must be empty or greater than START_DATE
 SUPPLIER_ROW_ID - numerical field

This API returns an event_id to the calling application, which can be used to check the status of the upload, and once it is finished The EVENT ID can be used to fetch any records that had errors.

Authorization Header:

username@apikey:password

Example Request Body:

```
{
  "neg_trips":[
    {
      "supplier_row_id":"1231231",
      "iss_agency":"0003",
      "start_date":"11/22/2015",
      "end_date":"11/22/2015",
      "cf_license_nbr":"999999"
    },
    {
      "supplier_row_id":"12312331",
      "iss_agency":"0003",
      "start_date":"11/23/2015",
      "end_date":"11/23/2015",
      "cf_license_nbr":"999999"
    }
  ]
}
```

Row response

.row[n].	
event_id	the unique job-id /event_id for this upload assigned by the system
upload_date	when the upload was received
process_date	When the upload was processed. This might be some time after the user submitted it
total_count	count of the neg_trips for this event_id
errored_count	count of rows that had an error of some kind. If the value is more than 0, NEG_UPLOAD_ERRORS should be called to get the details
.status	
.description	

NEG_TRIPS_UPLOAD_ERRORS - fetch a list of errors from negative reports that didn't load

Parameters:

P_EVENT_ID id from previous neg_upload command

Description:

Fetch any errors in the records from this event id. Returned as a JSON encoded object. If no rows had errors there will be no data, and status record will indicate zero errors

Authorization Header:

username@apikey:password

Example Request Body:

```
{  
  "p_event_id":"54321"  
}
```

Rows response

If there are no errors a status response of OK is returned, with a no rows. In this case the .rows.length property is zero.

The format returned is has 3 rows with addition of an ERR_DESC field at the start. That is used to hold the explanation of what is wrong with the record. The supplier_row_id is a row identifier that helps reference the original row passed by neg_upload. Event_id is the event id number given by neg_upload identifying the upload.

```
row[n].ERR_DESC  
      SUPPLIER_ROW_ID  
      EVENT_ID  
  
      .status  
      .description
```

LOOKUP_LIST - fetch a set of rows for a specified lookup table

Parameters:

P_PARTNER char, required, no wildcards. This is 4 character representation of the partner to be checked.

P_LISTNAME char, required, no wildcards. Name of the list being requested. More lists may be added with time. As of this publishing, the list is

- **GEARS** – returns list of rows with 5 columns. (code, description, category code, category description, lma flag, in-use flag)
- **GEAR_ATTRIBUTES** – returns a list of rows with 10 columns (attribute_code, attribute_name, attribute_description, required, category_code, units [array], options[array], order_seq, in-use). Unit array contains unit_code, unit_name, in-use flag.
- **CATCH_SOURCE** – returns a list of rows with 3 columns (source_code, source_description, in-use)
- **DISPOSITION**- returns list of rows with 5 columns. (code, description, category code, category description, trip type, in-use flag)
- **AREASFISHED** – returns a list of rows with 10 columns (area_code, area_name, sub_area_code, sub_area_name, local_area_code, local_area_name, state, area_display_name, waters, in-use flag)
- **SPECIES** - returns list of rows with 14 columns. The in_use indicator indicates whether a given row is currently active. Value returned is 'Y' when active and 'N' when no longer active.
- **TARGET_SPECIES** - returns list of rows with 3 columns (species_itis, species_name, in_use). The in_use indicator indicates whether a given row is currently active. Value returned is 'Y' when active and 'N' when no longer active.
- **TRIP_ACTIVITY** – returns list of rows with 4 columns (activity_type, activity_description, trip_type, in_use flag).
- **DEALERS** – returns a list of rows with 4 columns (corporate_name, full_name, license_nbr, agency).
- **PARTICIPANTS**– returns list of users and permits for a given partner. Each row has 3 columns. (ID, Name, License Nbr, in-use flag). All the licenses numbers returned are implicitly associated with the requested partner.
- **PORTS** – returns list of ports. Each row has 8 columns (port, fips_place_name, fips_county_name, state_postal, comm_flag, for_hire_hb_flag, for_hire_ch_flag, rec_flag).

Flag name	Description
comm_flag	commercial flag
for_hire_ch_flag	for hire charter flag
for_hire_hb_flag	for hire head boat flag
rec_flag	recreational flag

- **VESSELS**– returns list of vessels and associated permits. Each row has 7 columns. vessel id (ID), Name, registering state (state), coast guard number, state registration number, vessel permit number (permit), in-use flag). At this time the P_PARTNER field is ignored.
- **SWIPE_CARDS**– returns list of swipe cards for a given partner. Each row has 3 columns. (card number, License Nbr, in-use flag). All the licenses numbers returned are implicitly associated with the requested partner.
- **USER_NOTIFICATIONS**– returns list of HTML formatted messages from a partner to all users with a permit issued by that partner. and permits for a given partner. Each row has 3 columns. (m_id, Message,start_date, exp_date). Only non-expired messages are returned.
- **PARTNER_OPTIONS**– returns list of allowable options for the partner. For each option there is a value of Required (R), Optional (O) and Not Available (N). Current valid partner options are shown below.

CODE	Description
ECO	eTRIP - collect trip economic fields
MTR	eDR/mobile Only - automatically generate Trip Report
SUB	eTRIP - in efforts, display only SUB AREA
LOC	eTRIP - in efforts, display only LOCAL AREA CODE
LAC	use only Local Area Fished
PVT	partner will also supply vtr#
NDR	eTRIP - automatically create Dealer Report

P_LASTDATE (*optional*) changes made since a given date. Format is MM/DD/YYYY hh24:mi:ss. So a representation of Dec 27th 2010 2:45p.m. would be given as 12/27/2010 14:45:00 While optional, the p_lastdate is highly recommended to increase performance since it only returns a smaller list when a minimum date is used.

P_LISTTYPE (*optional*) char, no wildcards. This specifies the list being requested. Allowed values are EDR, ETRIP, ETRIPFORHIRE and ETRIPCOMMERCIAL. If no values is provided, ETRIPCOMMERCIAL is used as the default. The P_LISTTYPE option is available for all lists with the exception of PORTS and DEALERS.

Authorization Header:

@apikey

Example Request Body:

```
{
  "p_partner":"0023",
  "p_listname":"GEARS",
  "p_listtype":"EDR",
  "p_lastdate":"12/27/2010 14:45:00 "
}
```

Description:

Fetch a list of valid values for a specified list noted in p_listname above. Returned as a JSON encoded object. If no rows exist, there will be no data, and status record will indicate zero errors

Rows response

If there are no errors a status response of OK is returned. The row response varies based on the type of P_LISTNAME that was passed in.

<p>For gears, the format is as follows.</p> <pre>row[n].CODE DESCRIPTION CATEGORY_CODE CATEGORY_NAME LMA IN_USE</pre>	<p>For catch source, the format is as follows.</p> <pre>row[n]. CODE DESCRIPTION IN_USE .status .description</pre>
---	--

<p>.status .description</p> <p>For disposition, the format is as follows.</p> <pre>row[n].CODE DESCRIPTION CATEGORY_CODE CATEGORY_NAME TRIP_TYPE IN_USE</pre>	<p>For species, the format is as follows.</p> <pre>row[n].SPECIES_QC_ID SPECIES_ITIS SPECIES_NAME UOM GRADE GRADE_DESCRIPTION MARKET MARKET_DESCRIPTION MIN_PRICE MAX_PRICE HMS SHARK AREA IN_USE</pre>
<p>.status .description</p> <p>For <i>areasfished</i>, the format is as follows.</p> <pre>row[n].CODE DESCRIPTION CATEGORY_CODE CATEGORY_NAME WATERS IN_USE</pre>	<p>.status .description</p> <p>For participants, the format is as follows.</p> <pre>row[n].ID NAME LICENSE_NBR LICENSE_TYPE IN_USE</pre>
<p>.status .description</p> <p>For vessels, the format is as follows.</p> <pre>row[n].ID NAME STATE COAST_GUARD STATE_REG PERMIT IN_USE</pre>	<p>.status .description</p> <p>For <i>swipe_cards</i>, the format is as follows.</p> <pre>row[n].CARD_NBR LICENSE_NBR IN_USE</pre>
<p>.status .description</p> <p>For <i>target_species</i>, the format is as follows.</p> <pre>row[n]. SPECIES_ITIS SPECIES_NAME IN_USE</pre>	<p>.status .description</p> <p>For <i>gear_attributes</i>, the format is as follows.</p>

<p>.status .description</p> <p>For trip_activity, the format is as follows.</p> <p>row[n].ACTIVITY_TYPE ACTIVITY_DESCRIPTION TRIP_TYPE IN_USE</p> <p>.status .description</p> <p>For ports, the format is as follows.</p> <p>row[n].PORT FIPS_PLACE_NAME FIPS_COUNTY_NAME STATE_POSTAL COMM_FLAG FOR_HIRE_HB_FLAG FOR_HIRE_CH_FLAG REC_FLAG</p> <p>.status .description</p>	<p>attribute_code, attribute_name, attribute_description, category_code, units [array], in-use</p> <p>row[k].ATTRIBUTE_CODE ATTRIBUTE_DESC ATTRIBUTE_NAME REQUIRED CATEGORY_CODE TYPE IN_USE ORDER_SEQ UNITS [m].[CODE] [NAME] OPTIONS [m].[code] [display name]</p> <p>.status .description</p> <p>For dealers, the format is as follows.</p> <p>row[n].CORPORATE_NAME FULL_NAME LICENSE_NBR AGENCY</p> <p>.status .description</p>
---	--

LOOKUP_LIST_DATE_CHANGED - fetch the last date a specified lookup table was changed

Parameters:

P_PARTNER char, required, no wildcards. This is 4 character representation of the partner to be checked.

P_LISTNAMES array of list names of the list whose last date change is being requested. More lists may be added with time. As of this publishing, the list is

- *GEARS*
- *GEAR_ATTRIBUTES*
- *DISPOSITION*
- *AREASFISHED*
- *SPECIES*
- *CATCH_SOURCE*
- *PARTICIPANTS*
- *VESSELS*
- *USER_NOTIFICATIONS*
- *SWIPE_CARDS*
- *TRIP_ACTIVITY*
- *PORTS*
- *DEALERS*

Description:

Fetch the last date a specified lookup table, noted in `p_listnames`, was changed. The value is returned in the format of MM/DD/YYYY hh24:mi:ss. So a representation of Dec 27th 2010 2:45p.m. would be given as 12/27/2010 14:45:00
Returned as a JSON encoded object. If no date exist, there will be result will be empty

Authorization Header:

@apikey

Example Request Body:

```
{
  "p_partner":"0023",
  "p_listnames":["GEARS","VESSELS"]
}
```

Rows response

If there are no errors a status response of OK is returned.

For all valid values in `P_LISTNAMES`, the format is as follows.

```
row[n].last_date
      list
.status
.description
```

Example Return Body:

```
{
  "row": [
    {
      "list": "GEARS",
      "last_date": "09/26/2018 16:55:04"
    },
    {
      "list": "VESSELS",
      "last_date": "09/01/2016 12:44:01"
    }
  ],
  "status": "OK",
  "description": "Date(s) Found"
}
```

PERMIT_COMPLIANCE - checks the current compliance status of a dealer permit**Parameters:**

P_LICENSE_NBR alphanumeric , required. The license number of the permit number to be checked.

P_PARTNER The partner id associated with the license number given above.

Description:

The calling app will send a list of three variables above to identify the individual checking and the permit being checked.

Authorization Header:

username@apikey:password

Example Request Body:

```
{
  "p_license_nbr":"54321",
  "p_partner":"0023"
}
```

Row response

a status of OK is also returned for a successful check.

.compliant	a flag (Y or N) to identify whether a given permit is compliant
.last_record_date	the date of the last report entered into the system
.last_record_type	the type of record last entered (P/N) for positive or negative report
.non_compliance_dates	A list of dates that the dealer is not compliant in the format MM/DD/YYYY that are delimited by a semi-colon. Ex. "09/10/2015;09/11/2015"
.status	
.description	

REPORTS_UPLOAD - send a set of rows to safis

Parameters:

REPORTS table of dealer report and landing records (see description below)

Description:

The calling app will compose 2 tables of dealer reports and landings associated with those reports to upload. Note that since they are submitted as the JSON body of an http POST request they are all char data even if the underlying datatype is date or number.

The dealer report information is submitted as a collection of rows with each value denoting the type and column number it belongs.

Below is a list of the valid dealer report columns expected to make a complete row:

CF_LICENSE_NBR	STATE_REG_NBR
CF_PARTNER_ID	VTR_NBR
LANDING_DATE	DEA_LICENSE_NBR
TIME_LANDED	DEA_PARTNER_ID
TRIP_START_DATE	OBSERVER_LOG_ID
TRIP_START_TIME	HMS_LATE_REPORT
DATE_OF_PUR	SUPPLIER_DR_ID
PORT	SUBMIT_METHOD
COAST_GUARD_NBR	

Below is a list of valid landings columns expected to make a complete row. Note that the SUPPLIER_DR_ID denotes which dealer report it belongs.

REPORTED_QUANTITY	TOTAL_GEAR	ADDITIONAL_UNIT
DOLLARS	AREA_FISHED	HMS_FINS_ATTACHED
DISPOSITION_CODE	SUB_AREA_FISHED	HMS_EXPLANATION
GRADE_CODE	LOCAL_AREA_CODE	HMS_AREA_CODE
UNIT_MEASURE	CATCH_SOURCE	HMS_SALE_PRICE
SPECIES_ITIS	TIME_OF_HARVEST	FISHING_HOURS
MARKET_CODE	TIME_OF_ICING	HOURS_DAYS
PRICE	TEMP_AT_RECEIVING	SUPPLIER_LANDING_ID
GEAR_CODE	TEMP_UNIT (F or C)	
GEAR_QUANTITY	ADDITIONAL_COUNT	

This API returns an event_id to the calling application, which can be used to check the status of the upload, and once it is finished The EVENT ID can be used to fetch any records that had errors.

Authorization Header:

username@apikey:password

Example Request Body:

```
{
  "reports":[
    {"supplier_dr_id":"1231231",
     "state_reg_nbr":"SMUE939",
     "dea_license_nbr":"773737",
     "landing_date":"11/22/2015",
     "cf_license_nbr":"999999",
     "landings":[
       { "species_itis":"172409", "reported_quantity":"234", "dollars":"80.22",
         "grade_code":"01", "supplier_landing_id":"1202"},
       { "species_itis":"173408", "reported_quantity":"234", "dollars":"80.22",
         "grade_code":"01", "supplier_landing_id":"1203"}
     ]
    }
  ]
}
```

Row response

a status of OK is returned, with a single row that has same format at the UPLOAD_LIST API, but for just this one upload

.row[n].	
event_id	the unique job-id /event_id for this upload assigned by the system
upload_date	when the upload was received
process_date	when the upload was processed, this might be some time after the user submitted it
total_report_count	count of the supplier_dr_id for a single event_id

total_landing_count	count of the supplier_landing_id for a single event_id
errorred_report_count	count of the reports with at least 1 error for a single event_id
errorred_landing_count	count of rows that had an error of some kind. If the value is more than 0, UPLOAD_GET should be called to get the details
.status	
.description	

REPORTS_UPLOAD_ERRORS - fetch a list of errors from rows that did not load

Parameters:

P_EVENT_ID id from previous upload_put or upload_list command

Description:

Fetch any errors in the records from this event id. Returned as a JSON encoded object. If no rows had errors there will be no data, and status record will indicate zero errors

Authorization Header:

username@apikey:password

Example Request Body:

```
{  
  "p_event_id":"54321"  
}
```

Rows response

If there are no errors a status response of OK is returned, with a no rows. In this case the .rows.length property is zero.

The format returned is has 4 rows with addition of an ERR_DESC field at the start. That is used to hold the explanation of what is wrong with the record. The supplier_landing_id and supplier_dr_id combine to form a row identifier that helps reference the original row passed by reports_upload. Note that supplier_landing_id will be empty if the error is on the dealer_report information.

```
row[n].ERR_DESC  
      SUPPLIER_DR_ID  
      SUPPLIER_LANDING_ID  
      EVENT_ID  
  
      .status  
      .description
```

PERMIT_ENDORSEMENTS - fetch a list of endorsements for a given a partner

Parameters:

P_PARTNER char, required, no wildcards. This is 4 character representation of the partner to be checked.

P_LISTNAME name of the list being requested. More lists may be added with time. As of this publishing, the list is

- **VESSELS_ON_LICENSE**– returns list of rows with 6 columns. (vessel ID, name, coast guard #, state reg #, license #, in use flag)
- **ENDORSEMENTS_ON_LICENSE** - returns list of rows with 3 columns (license_nbr, endorsement type, in_use flag). The in_use indicator indicates whether a given e is currently active. Value returned is 'Y' when active and 'N' when no longer active.
- **SPECIES_BY_ENDORSEMENT** - returns list of rows with 6 columns (name, species_itis, endorsement type, health_dept, area, in_use flag). The HEALTH_DEPT field has a value of Y or N indicating whether this species requires the dept. of health fields. The AREA field has a value of Y or N indicating whether the area fished is required for this species. The in_use indicator indicates whether a given row is currently active. Value returned is 'Y' when active and 'N' when no longer active.
- **SPECIES_BY_LICENSE** - returns list of rows with 6 columns (name, species_itis, license_type, health_dept, area, in_use flag). The HEALTH_DEPT field has a value of Y or N indicating whether this species requires the dept. of health fields. The AREA field has a value of Y or N indicating whether the area fished is required for this species. The in_use indicator indicates whether a given row is currently active. Value returned is 'Y' when active and 'N' when no longer active.

P_LASTDATE (*optional*) changes made since a given date. Format is MM/DD/YYYY hh24:mi:ss. So a representation of Dec 27th 2010 2:45p.m. would be given as 12/27/2010 14:45:00 While optional, the p_lastdate is highly recommended to increase performance since it only returns a smaller list when a minimum date is used.

P_LICENSE (*optional*) filter the endorsements with a given license number. This field must be omitted to get the full list.

Description:

Fetch a list of valid values for a specified list noted in p_listname above. Returned as a JSON encoded object. If no rows exist, there will be no data, and status record will indicate zero errors

Authorization Header:

@apikey

Example Request Body:

```
{
  "p_partner":"0023",
  "p_listname":"VESSELS_ON_LICENSE",
  "p_license":"123456",
  "p_lastdate":"12/27/2010 14:45:00 "
}
```

Rows response

If there are no errors a status response of OK is returned. The row response varies based on the type of P_LISTNAME that was passed in.

For vessels_on_license, the format is as follows.

```
row[n].VESSEL_ID  
      NAME  
      COAST_GUARD  
      STATE_REG  
      LICENSE_NBR  
      IN_USE  
.status  
.description
```

For endorsements on license, the format is as follows.

```
row[n]. LICENSE_NBR  
      ENDORSEMENT_TYPE  
      IN_USE
```

For species_by_endorsement, the format is as follows.

```
row[n].ENDORSEMENT_TYPE  
      SPECIES_ITIS  
      NAME  
      HEALTH_DEPT  
      IN_USE  
.status  
.description
```

For species_by_license, the format is as follows.

```
row[n].LICENSE_TYPE  
      SPECIES_ITIS  
      NAME  
      HEALTH_DEPT  
      IN_USE  
.status  
.description
```

NEG_REPORTS_UPLOAD - send a set of negative dealer reports to safis

Parameters:

NEG_REPORTS table of neg_reports records (see description below)

Description:

The calling app will compose a table of negative dealer report records to upload, note that since they are submitted as the body of an http POST request they are all char data even if the underlying datatype is date or number.

The data values are submitted as a collection of rows with each value denoting the type and column number it belongs.

Below is a list of the valid columns expected to make a complete row. All fields but the end date are required:

DEA_LICENSE_NBR - (valid alphanumeric field)
 ISS_AGENCY - (valid alphanumeric field)
 START_DATE - (MM/DD/YYYY)
 END_DATE - (MM/DD/YYYY) must be empty or greater than START_DATE
 SUPPLIER_ROW_ID - numerical field

It returns an event_id to the calling application, which can be used to check the status of the upload, and once it is finished The EVENT ID can be used to fetch any records that had errors.

Authorization Header:

username@apikey:password

Example Request Body:

```
{
  "neg_reports":[
    {
      "supplier_row_id":"1231231",
      "iss_agency":"0003",
      "start_date":"11/22/2015",
      "end_date":"11/22/2015",
      "dea_license_nbr":"999999"
    },
    {
      "supplier_row_id":"12312331",
      "iss_agency":"0003",
      "start_date":"11/23/2015",
      "end_date":"11/23/2015",
      "dea_license_nbr":"999999"
    }
  ]
}
```

Row response

a status of OK is returned, with a single row that has same format at the UPLOAD_LIST API, but for just this one upload

.row[n].event_id the unique event_id for this upload assigned by the system
 upload_date when the upload was processed,
 this might be some time after the user submitted it
 process_date the date of processing completion
 total_count count of the total reports entered for this event. Note that
date range counts as one.
 errorred_count count of rows that had an error of some kind. If the value is
 more than 0, NEG_REPORTS_UPLOAD_ERRORS should be called to
 get the details

.status
.description

NEG_REPORTS_UPLOAD_ERRORS - fetch a list of errors from negative dealer reports that didn't load

Parameters:

P_EVENT_ID id from previous neg_upload command

Description:

Fetch any errors in the records from this event id. Returned as a JSON encoded object. If no rows had errors there will be no data, and status record will indicate zero errors

Authorization Header:

username@apikey:password

Example Request Body:

```
{  
  "p_event_id":"12345"  
}
```

Rows response

If there are no errors a status response of OK is returned, with a no rows. In this case the .rows.length property is zero.

The format returned is has 3 rows with addition of an ERR_DESC field at the start. That is used to hold the explanation of what is wrong with the record. The supplier_row_id is a row identifier that helps reference the original row passed by neg_upload. Event_id is the event id number given by neg_upload identifying the upload.

```
row[n].ERR_DESC  
      SUPPLIER_ROW_ID  
      EVENT_ID  
  
      .status  
      .description
```

PARTICIPANTS - check for users with the given information

This call retrieves a list of participants in the SAFIS system with the supplier user identifiable information. If one does not exist, they are created with the information supplied and their ACCSP id is returned.

Parameters for each participant(user) requested:

SUPPLIER_USER_ID	char, required, no wildcards
EMAIL	char, required, no wildcards
FIRST_NAME	char, required, no wildcards
LAST_NAME	char, required, no wildcards
BIRTH_DATE	char, required, Format (MM/DD/YYYY)
ISS_AGENCY	char, required, no wildcards
LICENSE_NBR	char, (optional) , no wildcards.
ADDRESS_1	char, (optional) , no wildcards.
ADDRESS_2	char, (optional) , no wildcards.
CITY	char, (optional) , no wildcards.
STATE	char, (optional) , no wildcards.
COUNTY	char, (optional) , no wildcards.
POSTAL_CODE	char, (optional) , no wildcards.
PHONE_NBR	char, (optional) , no wildcards.

Description

The API receives a list of 1 or more rows of user information containing at least all of the required fields above. The SAFIS system will return a user participant id and permit information to be used in the TRIPS_UPLOAD api call.

Authorization Header:

@apikey

Example Request Body:

```
{
  "participants": [
    {
      "supplier_user_id": "1111",
      "email": "mchristopher@elementalmethods.com",
      "first_name": "Michael",
      "last_name": "Christopher",
      "birth_date": "12/21/1982",
      "iss_agency": "0018",
      "license_nbr": "123457",
      "address_1": "5605 NORTH MACARTHUR BLVD",
      "address_2": "Suite 1000",
      "city": "Irving",
      "state": "TX",
      "county": "091",
      "postal_code": "75053",
      "phone_nbr": "972-819-2101"
    },
    {
      "supplier_user_id": "1112",
      "email": "james@elementalmethods.com",
      "first_name": "Jamie",
      "last_name": "Caranza",
      "birth_date": "12/01/2021",
      "iss_agency": "0018",
      "license_nbr": "123457",
      "address_1": "706 NORTH GINES BLVD",
      "address_2": "Suite 2000",
      "city": "Irving",
      "state": "TX",
      "county": "922",
      "postal_code": "75053",
      "phone_nbr": "972-819-310122"
    }
  ]
}
```

}

Returns

```

participants[n].DATA
    .status
    .description
errors[n].DATA
    .supplier_user_id
    .messages [k]
    
```

Row Response

It returns a list of license numbers and corresponding agency for the user whose user type is passed in. A status of OK is returned, with a single row that has the username and the total count of licenses being returned. If all records were successfully processed, the errors array will be empty.

.participants[n].	
supplier_user_id	Unique identifier supplied by the requestor
safis_participant_id	SAFIS system participant id to be passed into PARTICIPANT_ID of TRIPS_UPLOAD API
safis_license_number	SAFIS License Number to be passed into CF_LICENSE_NBR of TRIPS_UPLOAD API
issuing_agency	Issuing Agency of the corresponding license number to be passed into ISS_AGENCY of TRIPS_UPLOAD API
.errors[m].	
supplier_user_id	The user id supplied in the input request
messages[k]	An array of error messages for this user information above
.status	Has a value of "OK" unless there was a file error, like invalid JSON. Does NOT refer to the status of each record.
.processed	The number of records received in the participants array
.description	

Example Response Body:

```
{
  "participants": [
    {
      "supplier_user_id": "1111",
      "safis_participant_id": 2133332,
      "safis_license_nbr": "1343422",
      "issuing_agency": "0018"
    }
  ],
  "errors": [ {
    "supplier_user_id": "1112",
    "messages": ["Phone number is not valid", "Future Birth Date Invalid" ] }
  ]
  "status": "OK",
  "processed": 2,
  "description": "Received Records count: 2"
}
```