

SAFIS Unified API

- Overview 2
- USER_LOGIN - log into the API service..... 5
- USER_LOGOUT - sign out of the API service 6
- USER_PERMITS - retrieve a list of permits for the user 7
- TRIPS_UPLOAD - send a set of trips rows to safis..... 9
- TRIPS_UPLOAD_ERRORS - fetch a list of errors from trip rows that did not load 13
- NEG_TRIPS_UPLOAD - send a set of negative trip reports to safis..... 14
- NEG_TRIPS_UPLOAD_ERRORS - fetch a list of errors from negative reports that didn't load 16
- LOOKUP_LIST - fetch a set of rows for a specified lookup table..... 17
- LOOKUP_LIST_DATE_CHANGED - fetch the last date a specified lookup table was changed 20
- TRIP_LOCATIONS - send a set of trip location data rows to safis 21
- 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

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/"
 - prod - "safis.accsp.org:8443/safis_prod/safis_prod/"
- 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

Most commands requesting or receiving confidential data will supply a unique P_SESSION parameter. This ID will be returned by the application login and stored in the web application. The only call that takes place outside of the context of a session is the login call that generates the session id in the first place.

It is important to recognize that this ID is tied to the device, not the user. Someone may well be logged in twice on two different devices at the same time!

It is also important to note that one device cannot be engaged in more than one session at the same time as far as the API is concerned. This is not normally an issue since the entire logon-update-query-logoff sequence usually takes place in a short period of time. We do not anticipate an individual session extending over long periods of time from a mobile application. If an application is connected for a longer time a timeout will disconnect the session if there is no activity for a while.

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

```
var mydata = JSON.stringify({"p_login": "paul", "p_password": "psmith", "p_apikey": "xxxxxx"});
jQuery.ajax
(
{ url: "https://safis.accsp.org:8443/safis_test/safist",
  data: mydata,
  dataType: "json",
  type: "POST",
  success: function(data,textStatus,jqXHR)
    {callback_success(data,jqXHR)},
  error: function(jqXHR,textStatus,errorThrown)
    {callback_error(jqXHR,textStatus,errorThrown)}}
)
);
```

Example JSON response

A typical response from this call would be:

```
{ "row": [{ "session_id": "4822163059"}  
  ]  
  "status": "OK",  
  "description": "user paul logged in"  
}
```

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 components are accessed in javascript like this:

```
Var s = data.status ;           // this will set s to "OK"  
Var d = data.description ;     // this will set d to "user paul logged in"  
Var a = data.row[0].sessionId; // this will set a to " 4822163059",  
                               // array elements are numbered from 0.
```

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.

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_LOGIN - log into the API service

This is the first command that must be sent before a sequence of commands. It validates that a user is known to the system and returns their session-key.

Parameters:

P_LOGIN char, required, no wildcards. The users login username.
P_PASSWORD char, required, no wildcards.
P_APIKEY char, required, no wildcards. The API key is unique to each developer. Please contact the ACCSP team to request an API key.

Example Request Body:

```
{  
  "p_login":"myusername",  
  "p_password":"mysecretpassword",  
  "p_apikey":"devkey1234566"  
}
```

Returns

JSON structure, with 1 row of data if login succeeded, 0 rows of data for a failure.

```
row[0].session_id  
  .status  
  .description
```

Description

Validates that the user exists and is allowed access.

Row Response

Returns the session-id. Returns it as a single row of data. Only ever returns one row.

row[0].SESSION_ID a numeric string, guaranteed to be unique for the current day.

USER_LOGOUT - sign out of the API service

This is the last command that should be sent to end a session of back-end commands.

This is the last command that should be sent to end a session of back-end commands.

Parameters:

P_SESSION number, 20-digit session id

Description:

A session will time-out on inactivity if a device is simply turned off. In development we expire sessions after ten minutes of inactivity. If a session is timed out in this way then the call will return the not-logged-on error.

A user can be connected from several different mobile devices at the same time, they are separate sessions and in no way related to each other.

Example Request Body:

```
{  
  "p_session":"123456789abcd"  
}
```

Row Response

JSON structure, with no rows of data.

```
row[ ]  
  .status  
  .description
```

Returns only a status. This is an example of a call that returns no rows.

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 session id.

Parameters:

P_SESSION number, 20-digit session id
 P_USERTYPE char, required, no wildcards.

Description

The session is passed in with the P_SESSION 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.

Example Request Body:

```
{
  "p_session":"12345678abc",
  "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 session 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.
in_use	A flag to indicate whether the permit is active or not: 'Y' or 'N' values.

.status
.description

TRIPS_UPLOAD - send a set of trips rows to safis

Parameters:

P_SESSION number, 20-digit session id
 P_PIN (optional) alphanumeric value. Might be required for GARFO operators.
 P_DATA 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:

Trip level information

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

Effort level information

DISTANCE	GEAR_SETS
IN_STATE	FISHING_HOURS
AREA_CODE	HOURS_DAYS
SUB_AREA_CODE	TOTAL_GEAR
LOCAL_AREA_CODE	GEAR_SIZE
LATITUDE	MESH_RING_LENGTH
LONGITUDE	MESH_RING_WIDTH
GEAR	STRETCH_SIZE
LMA	TARGET_SPECIES
GEAR_QUANTITY	AVG_DEPTH

SPECIES_ITIS	SALE_DISPOSITION_FLAG	DEA_ISS_AGENCY
DISPOSITION	DEALER_LICENSE_NBR	CATCH_SOURCE
MARKET_CODE	DATE_SOLD	CATCH_LATITUDE
GRADE_CODE	REPORTED_QUANTITY	CATCH_LONGITUDE
UNIT_OF_MEASURE	PRICE	SUPPLIER_CATCH_ID

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.

Example Request Body:

```
{  
  "p_session": "1234567",  
  "p_pin": "1222",  
  "trips": [{  
    "supplier_trip_id": "1231231",  
    "trip_start_date": "11/22/2015",  
    "cf_license_nbr": "999999",  
    "state": "MA",  
    "efforts": [{  
      "gear": "234",  
      "latitude": "-70.1213",  
      "distance": "1",  
      "area_code": "202",  
      "catches": [{  
        "species": "123131",  
        "quantity": "23",  
        "disposition": "100",  
        "grade_code": "01"  
      }  
    ]  
  }  
}
```

```
"species": "333333",  
  
"quantity": "12",  
  
"disposition": "050",  
  
"grade_code ": "23"  
  }  
}, {  
  
  "gear": "700",  
  
  "latitude": "-70.1212",  
  
  "distance": "2",  
  
  "area_code": "202"  
  }  
}  
}
```

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
errorred_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
.status	
.description	

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

Parameters:

P_SESSION number, 20-digit session id

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

Example Request Body:

```
{  
  "p_session":"123456789abc",  
  "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_catch_id is a row identifier that helps reference the original row passed by upload_put. Event_id is the event id number given by upload_put identifying the upload.

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

NEG_TRIPS_UPLOAD - send a set of negative trip reports to safis

Parameters:

P_SESSION number, 20-digit session id
 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.

Example Request Body:

```
{
  "p_session":"1234567",
  "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_SESSION number, 20-digit session id
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

Example Request Body:

```
{  
  "p_session":"123456789abc",  
  "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_APIKEY char, required, no wildcards. The API key is unique to each developer. Please contact the ACCSP team to request an API key.

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)
- **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 6 columns (species_name, uom, grade, market, min_price, max_price, 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.
- **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.
- **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.

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.

Example Request Body:

```
{
  "p_apikey":"123456789abc",
  "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 .status .description </pre> <p>For disposition, the format is as follows.</p> <pre> row[n].CODE DESCRIPTION CATEGORY_CODE CATEGORY_NAME TRIP_TYPE IN_USE .status .description </pre> <p>For <i>areasfished</i>, the format is as follows.</p> <pre> row[n].CODE DESCRIPTION CATEGORY_CODE CATEGORY_NAME WATERS IN_USE .status .description </pre>	<p>For catch source, the format is as follows.</p> <pre> row[n]. CODE DESCRIPTION IN_USE .status .description </pre> <p>For species, the format is as follows.</p> <pre> row[n].SPECIES_QC_ID SPECIES_NAME UOM GRADE MARKET MIN_PRICE MAX_PRICE IN_USE .status .description </pre> <p>For participants, the format is as follows.</p> <pre> row[n].ID NAME LICENSE_NBR LICENSE_TYPE IN_USE .status .description </pre> <p>For <i>swipe_cards</i>, the format is as follows.</p>
---	---

<p>For vessels, the format is as follows.</p> <pre>row[n].ID NAME STATE COAST_GUARD STATE_REG PERMIT IN_USE .status .description</pre>	<pre>row[n].CARD_NBR LICENSE_NBR IN_USE .status .description</pre>
---	---

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

Parameters:

P_APIKEY char, required, no wildcards. The API key is unique to each developer. Please contact the ACCSP team to request an API key.

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

P_LISTNAME name 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*
- *DISPOSITION*
- *AREASFISHED*
- *SPECIES*
- *CATCH_SOURCE*
- *PARTICIPANTS*
- *VESSELS*
- *SWIPE_CARDS*

Description:

Fetch the last date a specified lookup table, noted in `p_listname`, 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

Example Request Body:

```
{
  "p_apikey":"123456789abc",
  "p_partner":"0023",
  "p_listname":"GEARS"
}
```

Rows response

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

For all valid values of `P_LISTNAME`, the format is as follows.

`row[1].LAST_DATE`

```
.status
.description
```

TRIP_LOCATIONS - send a set of trip location data rows to safis

Parameters:

P_SESSION number, 20-digit session id
 P_EVENT_ID number, required. id from previous trips_upload command
 P_DATA table of trip location records (see description below)

Description:

The calling app will compose a table of trip location 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. are:

```
SUPPLIER_TRIP_ID
LATITUDE
LONGITUDE
TIMESTAMP
```

The timestamp format is YYYYMMDDhh24miss e.g. 20141225131755 for Dec 25th 2014 at 1:17:55p.m. This API call fetches the body of the request and loads the records into a table.

It returns an event_id to the calling application, which can be used as a reference for these records.

Example Request Body:

```
{
  "p_session":"1234567",
  "p_event_id":"1222",
  "locations":[
    {"supplier_trip_id":"1231", "latitude":"43.31238", "longitude":"-72.11428", "timestamp":"20141225131755"},
    {"supplier_trip_id":"4231", "latitude":"43.32238", "longitude":"-72.11628", "timestamp":"20141225131845"}
  ]
}
```

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

event_id	. the unique job-id /event_id for this upload assigned by the system
total_count	count of the locations processed for the event_id
.status	
.description	

PERMIT_COMPLIANCE - checks the current compliance status of a dealer permit

Parameters:

P_SESSION number, 20-digit session id

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.

Example Request Body:

```
{
  "p_session":"123456789abc",
  "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:

P_SESSION number, 20-digit session id
 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.

Example Request Body:

```
{
  "p_session":"1234567",
  "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

<code>errorred_report_count</code>	count of the reports with at least 1 error for a single <code>event_id</code>
<code>errorred_landing_count</code>	count of rows that had an error of some kind. If the value is more than 0, <code>UPLOAD_GET</code> 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_SESSION number, 20-digit session id
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

Example Request Body:

```
{  
  "p_session":"123456789abc",  
  "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_APIKEY char, required, no wildcards. The API key is unique to each developer. Please contact the ACCSP team to request an API key.

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

Example Request Body:

```
{
  "p_apikey":"123456789abc",
  "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:

P_SESSION number, 20-digit session id
 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.

Example Request Body:

```
{
  "p_session":"1234567",
  "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

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

Parameters:

P_SESSION number, 20-digit session id
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

Example Request Body:

```
{  
  "p_session":"123456789abc",  
  "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
```