Integrated security for an IP-connected world

Web-based API for the S2 NetBox™

API Version 3.1d

S2 Security Corporation
50 Speen St.
Suite 300
Framingham, MA 01701
www.s2sys.com
S2 Support: 508 663-2505

Version 3.1d, July 25, 2008

Table of Contents

Overview

The S2 NetBox is an integrated security management network appliance that supports access control, alarm monitoring, video surveillance, temperature monitoring, intercom, and administrative functions. Individual S2 NetBox units are typically deployed to manage security for single facilities, and may be networked for the purpose of building larger security networks.

Internally, the S2 NetBox deploys the familiar three-tier architecture in which the database tier uses PostgreSQL, the web server is based on GoAhead's embedded web server, and the business logic is provided by S2 Security Corporation. All components managed by the S2 NetBox are network-connected, and are managed through a web-based user interface.

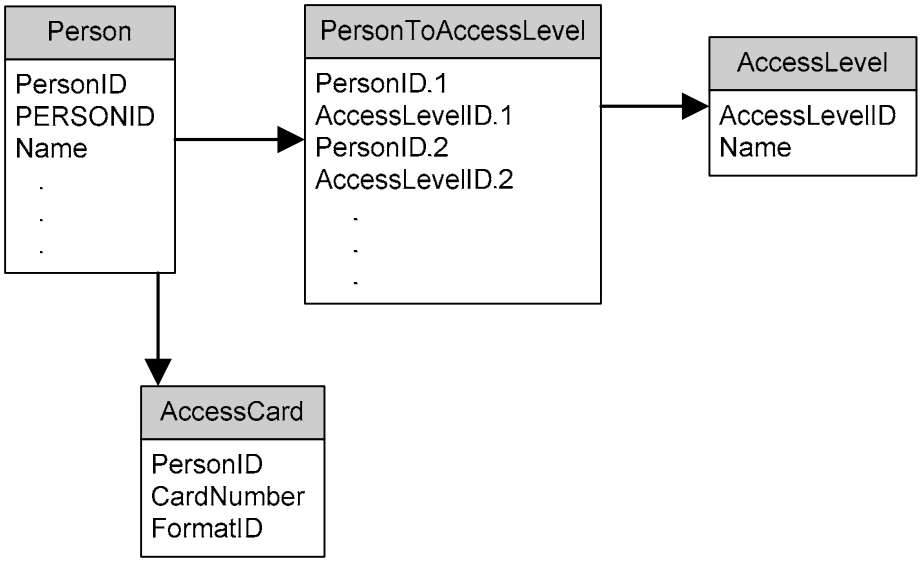
This document describes the API for the S2 NetBox that permits network-connected systems to perform various operations with the S2 NetBox under program control.

Database Architecture

The database server embedded in the S2 Network Controller (S2NC) is PostgreSQL, specifically licensed for this application. While the DBMS supports ODBC, to avoid conflict with possible future schema changes all database interaction should be accomplished through the API.

With regard to the access control subsystem, a fragment of the database schema is shown below. The database includes a table of "people", whose records act as container objects for attributes attached to people in real life. People are mapped to access levels which specify access privilege and to access cards, the credentials used for access control.

It is assumed that the access levels will be input into the system using the normal web user interface for the S2 NetBox, and that people and cards may be entered either through the web user interface or through the API. Likewise, card formats are entered through the standard web user interface. Note that because records of the Person table are basically container objects, a Person record must exist before an AccessCard or PersonToAccessLevel record is associated with it. The API provides a mechanism for accomplishing this by splitting the entry of the Person record from the credentials associated with that person. This is a two-step operation in the API, while the user interface encapsulates it as a single operation.



Calling the API

The API for the S2 NetBox is invoked by POSTing an HTTP message to the web server on the S2 Network Controller (S2NC). The message includes a single form parameter whose name is "APIcommand," and whose value is a blob of XML that contains the API command(s) described below.

API commands are accepted at address:

```
http://<netbox address>/goforms/nbapi
```

The IP port for these commands is the port at which the NetBox web server responds and is set with the SETUP > SITE SETTINGS > NETWORK CONTROLLER command. It defaults to port 80. By way of example, commands for a network controller located at 192.168.0.220 port 8080 go to `http://192.168.0.220:8080/goforms/nbapi`.

The XML API command passed to the S2 NetBox is wrapped in `<NETBOX-API>` tags. Within those tags are one or more[1] `<COMMAND>` blocks that contain individual API commands and a single `<MAC>` entry containing the computed message authentication code for the entire contents of the XML blob. Return information is passed back to the calling program from the web server, also in XML format.

The `<COMMAND>` block includes a command name attribute (`name=`) defined in the table below, a command number attribute[2] (`num=`) which serves to connect a given command with its response, and a block of parameters enclosed within `<PARAMS>` tags. Parameters are uniquely named, also as defined in the table below. Multiple `<COMMAND>` blocks can be included in a single blob of XML.

Each message sent by the caller contains a sequence number that is part of the message authentication code in the `<MAC>` tag. It is up to the caller to increment the sequence numbers; the S2 NetBox remembers the highest sequence number that is has seen, and will reject messages that have sequence numbers less than or equal to that number.[3]

XML tags that are not supported by the API but are syntactically valid XML are ignored. The structure of the XML blob is as follows:

```
<NETBOX-API>
   <COMMAND name=command-name1 num="1" dateformat="tzoffset">
      <PARAMS>
         <PARAM1> … value 1 … </PARAM1>
         <PARAM2> … value 2 … </PARAM2>
               .
               .
               .
      </PARAMS>
   </COMMAND>
   <COMMAND name=command-name2 num="2">
```

[1] The maximum number of `<COMMAND>` blocks that can be sent to the API is 32.

[2] The `num=` attribute is a number in the range from 1 to 32 (depending upon how many commands are provided. There can be no gaps in the numbering, so, for example, if there are two commands, they must be `num=1` and `num=2`.

[3] The system administrator has the option of turning off message authentication from the S2 NetBox user interface if there are other methods of data security in place. It is up to the caller to remember the highest sequence number transacted, and to always transact sequence numbers higher than the last one. If that becomes impossible, an option in the S2 NetBox user interface can be used to reset the sequence numbering.

```
      <PARAMS>
         <PARAM1> … value 1 … </PARAM1>
         <PARAM2> … value 2 … </PARAM2>
               .
               .
               .
      </PARAMS>
   </COMMAND>
         .
         .
         .
   <MAC> …authentication code as hex digits … </MAC>
</NETBOX-API>
```

The “dateformat” attribute on a command is optional. If it is supplied, dates are returned with an additional timezone offset included (see below for date format).

In response[4] to the API call, a blob of XML[5] is returned as follows, with each command generating a <RESPONSE> block that includes a “name=” attribute indicating the type of command to which it applies and a “num=” attribute indicating the instance, as follows:

```
<NETBOX>
   <RESPONSE command=command-name1 num=″1″>
      <CODE>  …response code text… </CODE>
      <DETAILS>  …response details… </DETAILS>
   </RESPONSE>
   <RESPONSE command=command-name2 num=″2″>
      <CODE>  …response code text… </CODE>
      <DETAILS>  …response details… </DETAILS>
   </RESPONSE>
      .
      .
      .
</NETBOX>
```

[4] No <MAC> is required for message responses because they are associated only with specific API requests.

[5] The maximum size of a response block is 8192 (8K) bytes.

Retrieving the version of the API

For versions 2.1 and newer, an XML command <GetAPIVersion> exists that will return <APIVERSION> with the appropriate version string. Versions of the API older than 2.1 do not recognize this command.

Therefore a request such as:

```
<NETBOX-API>
   <COMMAND name='GetAPIVersion' num="1">
   </COMMAND>
</NETBOX-API>
```

Will return (in the case of this version):

```
<NETBOX-API>
   <RESPONSE command='GetAPIVersion' num="1">
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <APIVERSION>2.2</APIVERSION>
      </DETAILS>
   </RESPONSE>
</NETBOX-API>
```

API Commands and Parameters

The API supports these commands:

- **AddPerson** – allows you to add a new Person to the system. This will return an error if the Person already exists (identified by a passed in PERSONID). If no PERSONID is supplied, one is created for the Person.
- **ModifyPerson** – allows you to modify an existing Person. PERSONID is required, and the call will fail if it does not match on an existing Person.

  This may be used to DELETE or UNDELETE a Person, as in the API*.*
- **EditPerson** – adds or modifies a person record to the database. *The use of this is deprecated in 2.5 – AddPerson and ModifyPerson are preferred*. This command associates a name and access level(s) with a person ID[6]. In order to associate a credential with the person, subsequent commands are issued to add credentials.

  The *EditPerson* command operates as follows:

  1. If a PERSONID is not supplied, then the data is entered into the Person table and the PERSONID of the new Person is returned to the caller or zero is returned if the add fails. In this case, the PERSONID will be assigned by the system and will be of the form "_nnnnnnn" (underscore followed by a sequence of digits).
  2. If a PERSONID is supplied by the caller and a Person with that ID does not exist in the database then the Person record is created and the PERSONID is returned to the caller or zero is returned if the add fails.

[6] A PERSONID is a text field supplied by the caller that is displayed in the "ID" field of the person record in the user interface. Typically this field holds an identifier of significance to the calling application's database such as an employee number.

3. If a PERSONID is supplied by the caller and a Person already exists in the database with that ID, then the record in the database is replaced by the data supplied by the caller and the PERSONID is returned or zero is returned if the operation failed.

EditPerson may also be used to Delete or "Undelete" a Person, as in the UI.

- **RemovePerson** – removes a person[7] and all associated credentials from the database.
- **GetPerson** – returns data about a Person.
- **SearchPersonData** – retrieves information about 1 or more people, based on various search criteria.
- **AddCredential** – adds a credential to a person already in the database.
- **RemoveCredential** – removes a credential from a person already in the database.
- **GetCardFormats** - retrieves a list of the names of the defined card formats.
- **AddAccessLevel** – add a new access level.
- **ModifyAccessLevel** – modify an existing access level.
- **DeleteAccessLevel** – delete an access level.
- **GetAccessLevel** – retrieve information about an access level.
- **GetAccessLevels** – returns a list of names of the valid access levels in the system.
- **PingApp** – heartbeat for the application. Used only for the caller to determine the health of the S2 NetBox.
- **GetAPIVersion** (2.1 and newer) – retrieves the current version of the NBAPI from the server.
- **GetAccessDataLog** – retrieve a data log from the history. The use of this is deprecated. Use *GetAccessHistory* instead.

  The *GetAccessDataLog* command operates as follows:

  1. The caller issues the *GetAccessDataLog* command passing the LOGID of the last data log retrieved from the API.
  2. If the caller does not know the LOGID of the last data log, then the caller passes zero to the *GetAccessDataLog* command and the command returns the last data log in the system with its LOGID.
  3. The caller adds one (1) to the LOGID of the last log received, and requests the data log with that LOGID. If no such data log exists (that is, if the LOGID passed was for the last log in the system), then the API will return a LOGID of zero.
- **GetAccessCardDetails** – returns recent card access events and related information given a CARDID and CARDFORMAT. *The use of this is deprecated. Use GetCardAccessDetails instead.*
- **GetCardAccessDetails** – returns recent card access events and related information given a CARDID and CARDFORMAT.
- **AddThreatLevel** – add in a new Threat Level.
- **ModifyThreatLevel** – modify an existing Threat Level.
- **EditThreatLevel** – add a new or edit an existing Threat Level *The use of this is deprecated in 2.5 – AddThreatLevel and ModifyThreatLevel are preferred.*

[7] RemovePerson actually *disables* the person record, leaving it in the database so that reports of history continue to function. Access levels and credential records are actually deleted, however.

- **RemoveThreatLevel** – delete a Threat Level.
- **SetThreatLevel** – change the Threat Level in the system.
- **AddThreatLevelGroup** – add a new Threat Level Group.
- **ModifyThreatLevelGroup** – modify threat level members in an existing threat level group.
- **EditThreatLevelGroup –** add a new or modify an existing Threat Levevl Group. *The use of this is deprecated in 2.5 – AddThreatLevelGroup and ModifyThreatLevelGroup are preferred.*
- **RemoveThreatLevelGroup** – remove a threat level group.
- **GetPortals** – returns list of portals and associated card readers defined for the NetBox.
- **GetAccessHistory** – returns a history of access activity, either for all users or for a particular access card (as a supplement to *GetAccessDataLog*, *GetCardAccessDetails*, and *GetAccessCardDetails*. *(GetAccessCardDetails is now deprecated: use GetCardAccessDetails instead.).*
- **GetTimeSpec** – returns a single time spec that is in the system.
- **GetTimeSpecs** – returns a list of time specs that are in the system.
- **AddTimeSpec** – add a new time spec into the system.
- **ModifyTimeSpec** – modify a time spec already in the system.
- **DeleteTimeSpec** – delete a time spec in the system.
- **GetTimeSpecGroup** – returns a single time spec group that is in the system.
- **GetTimeSpecGroups** – returns a list of time spec groups that are in the system.
- **AddTimeSpecGroup** – add a new time spec group into the system.
- **ModifyTimeSpecGroup** – modify a time spec group already in the system.
- **DeleteTimeSpecGroup** – Delete a time spec group from the system.
- **AddHoliday** – add a new holiday into the system.
- **ModifyHoliday** – modify an existing holiday.
- **DeleteHoliday** – delete an existing holiday.
- **GetHolidays** – return a list of holiday keys.
- **GetHoliday** – return a specific holiday.
- **AddReaderGroup** – add a new reader group to the system.
- **ModifyReaderGroup** – modify an existing reader group.
- **DeleteReaderGroup** – delete an existing reader group.
- **GetReaderGroups** – return a list of reader group keys.
- **GetReaderGroup** – return information for a specific reader group.
- **GetReaders** – return a list of readers in the system.
- **GetReader** – return information about a single reader.
- **AddPortalGroup** – add a new portal group.
- **ModifyPortalGroup** – modify an existing portal group.
- **DeletePortalGroup** - remove a portal group.

- **GetPortalGroups** – retrieve a list of Portal Groups.
- **GetPortalGroup** – retrieve information about a single portal group.

The table below details the available commands and their parameters.

Command Name	Parameter	Call / Return	Usage
AddPerson	LASTNAME	C	Text last name of person being added; this is a required field.
	FIRSTNAME	C	Text first name of person being added
	MIDDLENAME	C	Middle name of person
	NOTES	C	Notes field of person record
	EXPDATE	C	Expiration date for person record
	ACTDATE	C	Activation date for person record
	UDF1...UDF20	C	User-defined fields (20)
	ACCESSLEVELS	C	Block containing one or more access levels (maximum of 32) to be associated with the Person. See below for details.
	PICTURE	C	Optional: Picture data for the person being added.

	PICTUREEXT	C	Extension that describes the format of the picture data, eg, “jpg”. Required if PICTURE supplied: see above.: see previous parameter.
	PERSONID	C/R	“SUCCESS”, or “DUPLICATE” is returned as an ERROR if PERSONID matches an existing Person, and the Person record is not updated. If no PERSONID is supplied, one is created for the PERSON, and is returned in the result.
	ERRMSG	R	If the call returns an error as indicated by <CODE>, then <ERRMSG> contains a text description of the error condition. See *Examples* at the end of this document.
ModifyPerson	LASTNAME	C	Text last name of person being added; this is a required field.
	FIRSTNAME	C	Text first name of person being added
	MIDDLENAME	C	Middle name of person
	NOTES	C	Notes field of person record
	EXPDATE	C	Expiration date for person record
	ACTDATE	C	Activation date for person record
	UDF1...UDF20	C	User-defined fields (20)

	ACCESSLEVELS	C	Block containing one or more access levels (maximum of 32) to be associated with the Person. See below for details.
	PICTURE	C	Optional: Picture data for the person being modified.
	PICTUREEXT	C	Extension that describes the format of the picture data, eg “jpg”. Required if PICTURE supplied: see previous parameter.
	DELETED	C	“TRUE” or “FALSE.” Supplying “TRUE” is the same as a call to “RemovePerson.” Supplying “FALSE” to a person who has been deleted *undeletes* the Person.
	PERSONID	C	“SUCCESS”, or “FAIL” if PERSONID does not match an existing Person.
	ERRMSG	R	If the call returns an error as indicated by <CODE>, then <ERRMSG> contains a text description of the error condition. See *Examples* at the end of this document.
EditPerson (deprecated)	LASTNAME	C	Text last name of person being added; this is a required field.
	FIRSTNAME	C	Text first name of person being added
	MIDDLENAME	C	Middle name of person
	NOTES	C	Notes field of person record
	EXPDATE	C	Expiration date for person record

	ACTDATE	C	Activation date for person record
	UDF1...UDF20	C	User-defined fields (20)
	ACCESSLEVELS	C	Block containing one or more access levels (maximum of 32) to be associated with the Person. See below for details.
	PICTURE	C	Optional: Picture data for the person being added or modified.
	PICTUREEXT	C	Extension that describes the format of the picture data, eg "jpg". Required if PICTURE supplied: see previous parameter.
	DELETED	C	"TRUE" or "FALSE." Supplying "TRUE" is the same as a call to "RemovePerson." Supplying "FALSE" to a person who has been deleted *undeletes* the Person.
	PERSONID	C/R	"SUCCESS", "FAIL", or "DUPLICATE" as appropriate in <CODE> and the ID used by the S2 NetBox to identify the person added in <DETAILS>[8]. This parameter may be supplied by the caller or, if not supplied, will be assigned by the NetBox. In all cases, it is returned or zero if the operation fails.
	ERRMSG	R	If the call returns an error as indicated by

[8] In this command, DUPLICATE is a successful return indicating that the record was updated rather than inserted. A return of SUCCESS indicates an inserted record. (No reference in text to this footnote – actually, it's there under <DETAILS> above.)

			<CODE>, then <ERRMSG> contains a text description of the error condition. See *Examples* at the end of this document.
RemovePerson	PERSONID	C	ID of person to be removed (along with all of his cards) from the S2 NetBox
	result	R	"SUCCESS" or "FAIL" as appropriate returned in <CODE>.
GetPerson	PERSONID	C	PERSONID for the Person whose record the caller wishes to have returned.
	Result	R	"SUCCESS" or "FAIL" returned as <CODE>. If successful, then <DETAILS> contains a block of XML describing the Person (see below).
SearchPersonData	PERSONID	C	If supplied, the single person matching the person id will be returned
	LASTNAME	C	If these or any other of the criteria are used, the set of people matching the restrictions will be returned
	FIRSTNAME	C	
	MIDDLENAME	C	
	UDF1...UDF20	C	User Defined Fields
	PICTUREURL	R	
	DELETED	C	"TRUE", "FALSE" or "ALL." If "ALL" is supplied, both DELETED and UNDELETED people are returned. If "TRUE" is supplied, only DELETED people are returned, if

			"FALSE" is supplied, only NOT DELETED People are returned. Defaults to "FALSE" (i.e. only returns not deleted people).
	STARTFROMKEY	C	Used in conjunction with NEXTKEY to retrieve the next set of matching people
	NEXTKEY	R	This is returned with a value of "-1" if there are no more people to return, or a specific value > 0 that can be used in the next call as the "STARTFROMKEY" value.
AddCredential	PERSONID	C	ID of person for whom the credential is to be added
	ENCODEDNUM[9]	C	Optional: card number that will be placed in the "number" field of the card. This is now preferred to CARDNUM, as it will generate the corresponding raw hexadecimal digits. If HOTSTAMP or CARDID are not supplied, this is also used as the Hot stamp #.
	HOTSTAMP	C	Optional: "friendly" card number that is sometimes different from the one used in the underlying card format. If ENCODEDNUM is not supplied, this is used as the Encoded # as well.
	CARDNUM	C	Optional: Encoded

[9] The value for ENCODEDNUM must be an integer that fits within the number of bits specified in the CARDFORMAT for that credential.

			card number passed as a text string of hexadecimal digits. This number represents the raw card data that will be inserted into the record without the benefit of any additional formatting. The use of this is now deprecated. The functionality here can be more easily accomplished with the ENCODEDNUM field.
	CARDID	C	Optional: -- same meaning as HOTSTAMP. Its use is now deprecated.
	CARDFORMAT	C	Text name of the format to be used to decode the card
	result	R	"SUCCESS", "FAIL", or "DUPLICATE" as appropriate, with the latter meaning that the card already exists, is returned in <CODE>.
RemoveCredential	PERSONID	C	ID of the person from whom the credential is to be removed
	ENCODEDNUM/ HOTSTAMP/ CARDNUM/ CARDID	C	One of these will be used to identify the card, together with the CARDFORMAT. See the distinction between these fields in AddCredential.
	CARDFORMAT	C	Name of the card format to be used in interpreting the CARDID. This is the CARDFORMAT used in entering the credential through the user interface of via *AddCredential*.

	Result	R	"SUCCESS", "FAIL", or "NOT FOUND" as appropriate is returned in <CODE>.
GetCardFormats[10]	Result	R	"SUCCESS" or "FAIL" returned in <CODE> and a list of card format names returned in <DETAILS> if successful (see below).
AddAccessLevel	ACCESSLEVELNAME	C	Name for access level
	ACCESSLEVELDESCRIPTION	C	Description
	READERGROUPKEY	C	Optional, reference to ReaderGroup for this access level. Either this or READERKEY must be supplied
	READERKEY	C	Optional. Reference to Reader for this access level.
	TIMESPECGROUPKEY	C	Time Spec Group reference for this Access Level.
	THREATLEVELGROUPKEY	C	Threat level group reference for this Access Level
	ACCESSLEVELKEY	R	If success, the reference for this access level
ModifyAccessLevel	ACCESSLEVELKEY	C	
	ACCESSLEVELNAME	C	Name for access level
	ACCESSLEVELDESCRIPTION	C	Description
	READERGROUPKEY	C	Optional, reference to ReaderGroup for this access level. Either this or READERKEY must be supplied
	READERKEY	C	Optional. Reference to Reader for this access level.
	TIMESPECGROUPKEY	C	Time Spec Group

[10] GetCardFormats has no parameters.

			reference for this Access Level.
	THREATLEVELGROUPKEY	C	Threat level group reference for this Access Level
DeleteAccessLevel	ACCESSLEVELKEY	C	
GetAccessLevel	ACCESSLEVELKEY	C	
	ACCESSLEVELNAME	R	Name for access level
	ACCESSLEVELDESCRIPTION	R	Description
	READERGROUPKEY	R	Optional, reference to ReaderGroup for this access level. Either this or READERKEY must be supplied
	READERKEY	R	Optional. Reference to Reader for this access level.
	TIMESPECGROUPKEY	R	Time Spec Group reference for this Access Level.
	THREATLEVELGROUPKEY	R	Threat level group reference for this Access Level
GetAccessLevels	WANTKEY	C	if “TRUE” supplied, numeric key is returned; if “FALSE” or no parameter, key name is returned.
	STARTFROMKEY	C	Optional: when WANTKEY is TRUE, used in conjunction with NEXTKEY to retrieve the next set of Access Level Keys.
	STARTFROMNAME	C	Optional: when WANTKEY is FALSE (or not specified), used in conjunction with NEXTNAME to retrieve the next set of Access Level Names.

	result	R	"SUCCESS" or "FAIL" returned as <CODE> and a list of access levels returned in <DETAILS> if successful (see below).
GetAPIVersion[11]	APIVERSION	R	Returns a string "2.1" indicating the major and minor version of the API that is currently supported by the NBAPI
GetAccessDataLog	LOGID	C	ID number of the last log retrieved or zero to return the last data log in the system. "SUCCESS", "FAIL", or "NOT FOUND" returned in <CODE>.
	Result	R	See description below for details of returned data from the Get data log command.

[11] GetAPIVersion has no parameters, returns a version string, and is not supported before version 2.1 on the server.

GetCardAccessDetails and GetAccessCardDetails (GetAccessCardDetails is now deprecated: use GetCardAccessDetails instead.)	ENCODEDNUM/ HOTSTAMP/ CARDNUM/ CARDID	C	One of these will be used to identify the card, together with the CARDFORMAT. See the distinction between these fields in AddCredential.
	CARDFORMAT	C	Name of the card format to be used in interpreting the CARDID. This is the CARDFORMAT used in entering the credential through the user interface of via *AddCredential*.
	MAXRECORDS	C	Optional: Maximum number of access history data logs to return. If omitted, the API will return the smaller of the number of records that match the request and the number of records that fill its 16K return buffer.
	OLDESTDTTM	C	Optional: do not retrieve any accesses older than this time.
	Result	R	See notes below.
AddThreatLevel	LEVELNAME	C	Name of threat level to add.
	SEQNUM	C	Optional: Display order for the threat level in NetBox user interface displays.
	COLOR	C	Optional: One of `White, Green, Blue, Yellow, Orange`, or `Red` as desired.
	Return	R	"SUCCESS" or "FAIL" returned as `<CODE>`.

ModifyThreatLevel	LEVELNAME	C	Name of threat level to edit.
	SEQNUM	C	Optional: Display order for the threat level in NetBox user interface displays.
	COLOR	C	Optional: One of `White, Green, Blue, Yellow, Orange`, or `Red` as desired.
	Return	R	"SUCCESS" or "FAIL" returned as `<CODE>`.
EditThreatLevel (deprecated)	LEVELNAME	C	Name of threat level to edit (if it exists) or add (if it does not exist). Note: to rename a threat level, use *RemoveThreatLevel* followed by *EditThreatLevel*.
	SEQNUM	C	Optional: Display order for the threat level in NetBox user interface displays.
	COLOR	C	Optional: One of `White, Green, Blue, Yellow, Orange`, or `Red` as desired.
	Return	R	"SUCCESS" or "FAIL" returned as `<CODE>`.
RemoveThreatLevel	LEVELNAME	C	Name of the threat level to remove.
	Return	R	"SUCCESS" or "FAIL" returned as
SetThreatLevel	LEVELNAME	C	Name of the threat level into which the system will be set.
	Return	R	"SUCCESS" or "FAIL" is returned in `<CODE>`.

AddThreatLevelGroup	LEVELGROUPNAME	C	Name of the threat level group to add.
	LEVELNAMES	C	Optional: Names of the threat levels to be added to this threat level group. See below for an example of the call.
	Return	R	“SUCCESS” or “FAIL” is returned in <CODE>.
ModifyThreatLevelGroup	LEVELGROUPNAME	C	Name of the threat level group to edit. Note that all members of the threat level group are replaced with each call to this function.
	LEVELNAMES	C	Optional: Names of the threat levels to be added to this threat level group. See below for an example of the call.
	Return	R	“SUCCESS” or “FAIL” is returned in <CODE>.
EditThreatLevelGroup (deprecated)	LEVELGROUPNAME	C	Name of the threat level group to edit or insert. Note that all members of the threat level group are replaced with each call to this function.
	LEVELNAMES	C	Optional: Names of the threat levels to be added to this threat level group. See below for an example of the call.
	Return	R	“SUCCESS” or “FAIL” is returned in <CODE>.
RemoveThreatLevelGroup	LEVELGROUPNAME	C	Name of the threat level group to remove from the system.
	Return	R	“SUCCESS” or “FAIL” is returned in <CODE>.

GetPortals	STARTFROMKEY	C	Used in conjunction with NEXTKEY to retrieve the next set of portals
	NEXTKEY	R	This is returned with a value of "-1" if there are no more people to return, or a specific value > 0 that can be used in the next call as the "STARTFROMKEY" value.
GetAccessHistory	STARTLOGID	C	Optional: use to start with a particular log ID. Generally used in conjunction with <NEXTLOGID> returned from a prior call
	AFTERLOGID	C	Optional: used to start after a particular log ID that was previously returned. Implies an order of "ASCENDING"
	ORDER	C	Optional: can be DESCENDING or ASCENDING. Defaults to DESCENDING (unless AFTERLOGID is supplied)
	MAXRECORDS	C	Optional: maximum # of ACCESSENTRY's returned in one call. If not supplied, the maximum returned in one call is limited to an internal buffer size.
	ENCODEDNUM/ HOTSTAMP/ CARDNUM/ CARDID	C	Optional: one of these will be used to identify a particular card, together with the CARDFORMAT. See the distinction between these fields in AddCredential.
	CARDFORMAT	C	Optional: name of the card format to be

			used in interpreting the card identifier information. This is the CARDFORMAT used in entering the credential through the user interface of via *AddCredential*.
	OLDESTDTTM	C	Optional: do not retrieve any accesses older than this time.
	Return	R	“SUCCESS” OR “NOT FOUND” returned in <CODE>
GetTimeSpec	TIMESPECKEY	C	Key for TimeSpec
GetTimeSpecs	STARTFROMKEY	C	Used in conjunction with NEXTKEY to retrieve the next set of time specs
	NEXTKEY	R	This is returned with a value of “-1” if there are no more timespecs to return, or a specific value > 0 that can be used in the next call as the “STARTFROMKEY” value.
AddTimeSpec	NAME	C	Name for timespec
	DESCRIPTION	C	Description
	STARTTIME	C	Start Time in HH:MM format
	ENDTIME	C	End Time in HH:MM format
	MONDAY	C	1 / 0
	TUESDAY	C	1 / 0
	WEDNESDAY	C	1 / 0
	THURSDAY	C	1 / 0
	FRIDAY	C	1 / 0
	SATURDAY	C	1 / 0
	SUNDAY	C	1 / 0
	HOLIDAYGROUPS	C	Any of the numbers “1,2,3”, separated by commas (e.g. “1,2”

	TIMESPECKEY	R	Key of newly created timespec
	return	R	“SUCCESS” or “FAIL” returned as <CODE>.
ModifyTimeSp**ec**	TIMESPECKEY	C	Key for time spec to modify
	NAME	C	Name for timespec
	DESCRIPTION	C	Description
	STARTTIME	C	Start Time in HH:MM format
	ENDTIME	C	End Time in HH:MM format
	MONDAY	C	1 / 0
	TUESDAY	C	1 /. 0
	WEDNESDAY	C	1 / 0
	THURSDAY	C	1 / 0
	FRIDAY	C	1 / 0
	SATURDAY	C	1 / 0
	SUNDAY	C	1 / 0
	HOLIDAYGROUPS	C	Any of the numbers “1,2,3”, separated by commas (e.g. “1,2”
DeleteTimeSpec	TIMESPECKEY	C	
GetTimeSpecGroup	TIMESPECGROUPKEY	C	Key for TimeSpecGroup
GetTimeSpecGroups	STARTFROMKEY	C	Used in conjunction with NEXTKEY to retrieve the next set of time spec groups
	NEXTKEY	R	This is returned with a value of “-1” if there are no more timespecs to return, or a specific value > 0 that can be used in the next call as the “STARTFROMKEY” value.
AddTimeSpecGroup	NAME	C	Name for timespec group
	DESCRIPTION	C	Description

	TIMESPECKEYS	C	List of keys for timespecs to be included in the group
	TIMESPECGROUPKEY	R	Key of newly created timespec group
ModifyTimeSpecGroup	TIMESPECGROUPKEY	C	Key for time spec group to modify
	NAME	C	Name for timespec
	DESCRIPTION	C	Description
	TIMESPECKEYS	C	List of keys for timespecs to be included in the group
DeleteTimeSpec	TIMESPECKEY	C	
AddHoliday	HOLIDAYNAME	C	
	HOLIDAYGROUPS	C	Any of 1,2, or 3
	STARTDATE	C	
	ENDDATE	C	
	HOLIDAYKEY	R	Key of holiday newly created holiday
ModifyHoliday	HOLIDAYKEY	C	
	HOLIDAYNAME	C	
	HOLIDAYGROUPS	C	Any of 1,2, or 3
	STARTDATE	C	
	ENDDATE	C	
DeleteHoliday	HOLIDAYKEY	C	
GetHoliday	HOLIDAYKEY	C	
	HOLIDAYNAME	R	
	HOLIDAYGROUPS	R	
	STARTDATE	R	
	ENDDATE	R	
GetHolidays	HOLIDAYS	R	List of HOLIDAYKEYs

AddReaderGroup	NAME	C	Name for reader group
	DESCRIPTION	C	Description
	READERKEYS	C	List of keys for readers to be included in the group
	READERGROUPKEY	R	Returned identity for new reader group
ModifyReaderGroup	READERGROUPKEY	C	Identifier for the reader group
	NAME	C	
	DESCRIPTION	C	
	READERKEYS	C	
DeleteReaderGroup	READERGROUPKEY	C	
GetReaderGroups	READERGROUPKEYS	R	Returns a list of reader groups
GetReaderGroup	READERGROUPKEY	C	
	NAME	R	
	DESCRIPTION	R	
	READERKEYS	R	List of reader keys
GetReaders		R	List of reader keys in the system
GetReader	READERKEY	C	
	NAME	R	
	DESCRIPTION	R	
AddPortalGroup	NAME	C	Name for portal group
	DESCRIPTION	C	Description
	UNLOCKTIMESPECGROUPKEY	C	Timespec group key for unlocking portals in this group
	THREATLEVELGROUPKEY	C	Reference to threat level group
	PORTALKEYS	C	List of keys for portals to be included in the group
	PORTALGROUPKEY	R	Returned identity for new portal group

ModifyPortalGroup	PORTALGROUPKEY	C	Identifier for the reader group
	NAME	C	
	DESCRIPTION	C	
	PORTALKEYS	C	
	UNLOCKTIMESPECGROUPKEY	C	Timespec group key for unlocking portals in this group
	THREATLEVELGROUPKEY	C	Reference to threat level group
DeletePortalGroup	PORTALGROUPKEY	C	
GetPortalGroups	PORTALGROUPKEYS	R	Returns a list of portal groups
GetPortalGroup	PORTALGROUPKEY	C	
	NAME	R	
	DESCRIPTION	R	
	PORTALKEYS	R	List of reader keys
	UNLOCKTIMESPECGROUPKEY	C	Timespec group key for unlocking portals in this group
	THREATLEVELGROUPKEY	C	Reference to threat level group

Access Levels in *EditPerson*

A Person record may have up to thirty two (32) access levels associated with it. The syntax below describes the access levels associated with a Person:

```
<ACCESSLEVELS>
   <ACCESSLEVEL>access level 1</ACCESSLEVEL>
   <ACCESSLEVEL>access level 2</ACCESSLEVEL>
          .
          .
          .
   <ACCESSLEVEL>access level 32</ACCESSLEVEL>
</ACCESSLEVELS>
```

Any access levels passed with the *EditPerson* API command replace all access levels in the Person record. That is, access levels must be passed as a group. It is not necessary, however, to pass empty access levels; any access levels not provided in an *EditPerson* command will be cleared by the API.

Date Formats

There are different rules for supplying dates as input, and the dates that are retrieved. All dates are in local time as of the time of the NetBox server.

Activation Date and Expiration Date Formats in *EditPerson*

The activation date (ACTDATE) and expiration date (EXPDATE), if provided, must be in the form:

```
YYYY-MM-DD HH:MM  or  YYYY-MM-DD
```

where the time is defaulted to 00:00 in the second case.

Returned Dates

Returned dates are shown in 2 forms. If the command does not have the 'dateformat="tzoffset"' attribute, the dates are returned as:

```
YYYY-MM-DD HH:MM:SS
```

If 'dateformat="tzoffset"' is supplied as an attribute to the command, then thereturned value is in the format:

```
YYYY-MM-DD HH:MM:SS +/-HHMM
```

Where, for example, EST timezone would be "-0500" (indicating -5 hours and 0 minutes offset from GMT).

Using the ENCODEDNUM, HOTSTAMP, CARDID and CARDNUM Parameters in *AddCredential*

Callers have the option of supplying an ENCODEDNUM, HOTSTAMP, CARDID or CARDNUM with the AddCredential and RemoveCredential calls. CARDID is simply a synonym for HOTSTAMP and will not be discussed further. CARDNUM represents the internal bit representation of a card as stored in the database, and its use is deprecated, although it will be described below.

ENCODEDNUM and HOTSTAMP correspond to the fields that show up in the NetBOX UI. They often appear as the same values. However, their meaning is quite different. The ENCODEDNUM that shows up in the UI (and here in the NBAPI) is an integer that is inserted into the card according to the bit representation of the card format.

For example, the standard Wiegand26 format that comes with the NetBox displays the following in the NB UI:

```
-FFFFFFF FNNNNNNN NNNNNNNN N-
```

The "-" represent bits that are ignored, the "FFFF" represents the facility code, and the "NNNN" represent bits where the ENCODEDNUM is stored.

HOTSTAMP, while it defaults to the same value as ENCODEDNUM, is often a number that appears on the credential itself as an externally visible integer and may be quite different than the ENCODEDNUM.

In the NBAPI, either will be used for both if only one is supplied.

The CARDNUM Parameter and Card Format in *AddCredential*

As stated before, use of the CARDNUM parameter is now deprecated.

The CARDNUM parameter is the data stream that is encoded on the credential. In the database, it is a 128-bit value stored in the form it takes as it is read from the credential. That means that it is the complete data stream as it appears in the credential, and padded with zeroes to the right. The CARDNUM value passed to the API includes enough zero bits (padded to the right) to make complete bytes. That value is passed as the ASCII equivalent[12] of the hexadecimal data that form those bytes. The API will further pad to the right with zeroes until the full 128 bits is reached.

By way of example, consider the following 26 bits as read off a typical Wiegand card and then padded to the right with zeroes to make an integral number of bytes:

`00000101 11000001 11011111 10 → 00000101 110000001 11011111 10000000`

This value is equivalent to `0x05C1DF80` (hex) or "`05C1DF80`" in ASCII. Note that this 8-character string represents 32 bits. Zeroes are added to the right to make 128 bits, resulting in the value "`05C1DF80000000000000000000000000`". The caller could have, of course, simply passed this fully padded value and obtained the same result.

The raw data read from a credential in real time by the S2 NetBox is interpreted using a set of rules defined by the *card format*. Card formats are defined in the S2 NetBox user interface with the SETUP > ACCESS CONTROL > CARD FORMATS command and a list of the card format names can be retrieved in the API using the *GetCardFormats* command.

The response to the GetCardFormats API command is similar to:

```
<NETBOX>
   <RESPONSE command="GetCardFormats" num="1">
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <CARDFORMATS>
            <CARDFORMAT>Wiegand26</CARDFORMAT>
            <CARDFORMAT>ISO34bit</CARDFORMAT>
         </CARDFORMATS>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

[12] Only upper case characters may appear in the ASCII-equivalent representation of the hexadecimal data.

Response to *GetPerson*

When the *GetPerson* command returns a <CODE> of "SUCCESS", then <DETAILS> contains a block of XML with data about the Person. The entire response appears as:

```
<NETBOX>
   <RESPONSE command="GetPerson" num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <PERSONID>ID of Person record</PERSONID>
         <FIRSTNAME>Person's first name</FIRSTNAME>
         <LASTNAME>Person's last name</LASTNAME>
         <UDF1>User Defined Field</UDF1>
         <UDF2>User Defined Field</UDF1>
         …
         <UDF20>User Defined Field</UDF20>
         <PICTUREURL>Filename for picture data</PICTUREURL>
         <DELETED>TRUE/FALSE</DELETED>
         <ACCESSLEVELS>
            <ACCESSLEVEL>access level 1</ACCESSLEVEL>
            <ACCESSLEVEL>access level 2</ACCESSLEVEL>
               .
               .
               .
            <ACCESSLEVEL>access level 32</ACCESSLEVEL>
         </ACCESSLEVELS>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Note that only the access levels currently assigned are returned, and if there are none assigned, then none are returned. Also note that the picture data file (returned as text between the PICTUREURL tags) is stored in the directory "/usr/local/s2/web/upload/pics" on the controller.

Calling *SearchPersonData*

When calling *SearchPersonData*, there are various options in making the call. If <PERSONID> is supplied in the call, then exactly one matching person is returned. In the other cases, the call retrieves the matching people.

In the boundary case, with no parameters supplied, all people on the NetBox are returned. If there are more people than can be returned in a single response, a <NEXTKEY> element is returned that allows the next call to be made. In this way, iterative calls can retrieve as many people as required, by keeping all the other parameters the same, and supplying a new <NEXTKEY> element for each successive call.

An example of a call to retrieve a single person is:

```
<NETBOX-API>
   <COMMAND name="SearchPersonData" num="1" dateformat="tzoffset">
      <PARAMS>
         <PERSONID>_3</PERSONID>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
```

```
</NETBOX-API>
```

An example of a (second) call to retrieve multiple people with restrictions, after the first response returned a <NEXTKEY> parameter is:

```
<NETBOX-API>
   <COMMAND name=”SearchPersonData” num=”1”>
      <PARAMS>
         <LASTNAME>Smith</LASTNAME>
         <STARTFROMKEY>32</STARTFROMKEY>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Response to *SearchPersonData*

When the *SearchPersonData* command returns a <CODE> of “SUCCESS”, then <DETAILS> contains a block of XML with data about one or more people. The entire response appears as:

```
<NETBOX>
   <RESPONSE command=”SearchPersonData” num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <PEOPLE>
            <PERSON>
               <PERSONID>ID of Person record</PERSONID>
               <FIRSTNAME>Person’s first name</FIRSTNAME>
               <LASTNAME>Person’s last name</LASTNAME>
               <UDF1>User Defined Field</UDF1>
               <UDF2>User Defined Field</UDF1>
               …
               <UDF20>User Defined Field</UDF20>
               <PICTUREURL>Filename for picture data</PICTUREURL>
               <DELETED>TRUE/FALSE</DELETED>
               …
               <ACCESSLEVELS>
                  <ACCESSLEVEL>access level 1</ACCESSLEVEL>
                  <ACCESSLEVEL>access level 2</ACCESSLEVEL>
                  …
               </ACCESSLEVELS>
               <ACCESSCARDS>
                  <ACCESSCARD>
                     <ENCODEDNUM>encoded #</ENCODEDNUM>
                     <HOTSTAMP>hot stamp #</HOTSTAMP>
                     <CARDFORMAT>name of card format</CARDFORMAT>
                     <DISABLED>1 or 0</DISABLED>
                  </ACCESSCARD>
                  <ACCESSCARD>
                     …
                  </ACCESSCARD>
               </ACCESSCARDS>
            </PERSON>
            <PERSON>
               …
```

```
            </PERSON>
         </PEOPLE>
         <NEXTKEY>internal key to pass into next request</NEXTKEY>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

The breakdown of information is as follows. As many people are returned as possible that match the query. If there is insufficient space in the internal buffer, <NEXTKEY> is returned as the parameter to use to retrieve the next set of people.

For each person, the list of access levels and access cards associated with that person are returned. If there are no access levels, none are returned. For the access cards, the newer <ENCODEDNUM> and <HOTSTAMP> attributes are returned (see AddCredential) in place of <CARDID> and <CARDNUM>.

Note that the picture data file (returned as text between the PICTUREURL tags) is stored in the directory "/usr/local/s2/web/upload/pics" on the controller.

Response to GetAccessLevels

When the *GetAccessLevels* command returns a <CODE> of "SUCCESS" then <DETAILS> contains a list of access level names currently defined in the system. The entire return would appears as follows:

```
<NETBOX>
   <RESPONSE command="GetAccessLevels" num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <ACCESSLEVELS>
            <ACCESSLEVEL>access level 1</ACCESSLEVEL>
            <ACCESSLEVEL>access level 2</ACCESSLEVEL>
               .
               .
               .
            <ACCESSLEVEL>access level 256</ACCESSLEVEL>
         </ACCESSLEVELS>
         <NEXTNAME>access level 257</NEXTNAME>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Note that only the access levels currently assigned are returned, and if there are none assigned, then none are returned.

If there is insufficient space in the internal buffer, either <NEXTNAME> or <NEXTKEY> (if WANTKEY=TRUE) is returned as the parameter to use to retrieve the next set of people.

Otherwise, </NEXTNAME>, or <NEXTKEY> with a value of "-1", are returned, respectively.

Response to the *GetCardAccessDetails* and *GetAccessCardDetails* Command

(GetAccessCardDetails is deprecated: use GetCardAccessDetails instead.)

The *GetCardAccessDetails* and *GetAccessCardDetails* API commands are used to return recent access events associated with a given credential as defined by that credential's CARDID and CARDFORMAT parameters. A typical call looks like:

```
<NETBOX-API>
   <COMMAND name="GetCardAccessDetails" num="1">
      <PARAMS>
         <ENCODEDNUM>3527</ENCODEDNUM>
         <CARDFORMAT>Code30</CARDFORMAT>
         <MAXRECORDS>3</MAXRECORDS>
      </PARAMS>
   </COMMAND>
</NETBOX-API>
```

In this example, the most recent 3 access events are requested for a credential with ENCODEDNUM 3527and a CARDFORMAT called "Code30." If the request succeeds (<CODE>SUCCESS</CODE>) then the following are returned:

- PERSONID – the external person ID associated with the person who owns the credential specified credential. This is the field in the person record labeled "ID #".
- DISABLED – 1 if the credential is currently marked disabled, and 0 otherwise.
- DTTM – The controller date and time associated with the first access data log that exists for the stated credential on the day the request is made to the API.
- PORTALNAME – Name of the portal associated with the first access of the calendar day referred to by the DTTM value.
- ACCESSES – a list of access data log records responsive to the request. Each data log returned is enclosed within <ACCESS>…</ACCESS> tags, and may include the following fields:
    - DTTM – The controller date and time associated with the data log.
    - NODEDTTM - The node date and time associated with the data log.
    - TYPE – Valid or invalid access. See the table on page 35 for details on the type codes that can be returned.
    - PORTALNAME – name of the portal associated with the access.
    - PORTALKEY – unique identifier for the portal that matches to the GetPortals query.
    - READERKEY – unique identifier for the reader that had the access
    - REASON – Reason code associated with a rejected access attempt. See the table on page 35 for details.

A sample might look like:

```
<NETBOX>
   <RESPONSE command="GetCardAccessDetails" num="1">
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <PERSONID>44886</PERSONID>
         <DISABLED>0</DISABLED>
         <ACCESSES>
            <ACCESS>
               <DTTM>2005-11-09 15:07:03</DTTM>
```

```
               <TYPE>1</TYPE>
               <PORTALNAME>Front Door</PORTALNAME>
               <PORTALKEY>30</PORTALKEY>
               <READERKEY>3</READERKEY>
               <LOGID>523</LOGID>
               <REASON></REASON>
            </ACCESS>
            <ACCESS>
               <DTTM>2005-11-09 11:38:15</DTTM>
               <TYPE>1</TYPE>
               <PORTALNAME>Garage</PORTALNAME>
               <PORTALKEY>41</PORTALKEY>
               <READERKEY>4</READERKEY>
               <LOGID>521</LOGID>
               <REASON></REASON>
            </ACCESS>
            <ACCESS>
               <DTTM>2005-09-15 10:02:57</DTTM>
               <TYPE>1</TYPE>
               <PORTALNAME>Main Gate</PORTALNAME>
               <PORTALKEY>23</PORTALKEY>
               <READERKEY>5</READERKEY>
               <LOGID>479</LOGID>
               <REASON></REASON>
            </ACCESS>
         </ACCESSES>
         <DTTM>2005-11-09 11:38:15</DTTM>
         <PORTALNAME>Garage</PORTALNAME>
         <PORTALKEY>41</PORTALKEY>
         <READERKEY>4</ READERKEY>
         <NEXTLOGID>450</NEXTLOGID>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Response to the *GetAccessDataLog* Command

Responses to the *GetAccessDataLog* command indicate the details of card access attempts, and have the form:

```
<NETBOX>
   <RESPONSE command="GetAccessDataLog" num=command-number>
      <CODE>SUCCESS|FAIL|NOT FOUND</CODE>
      <DETAILS>
         <LOGID>logid</LOGID>
         <PERSONID>ID of Person record</PERSONID>
         <READER>name of reader</READER>
         <READERKEY>unique identifier for reader</READERKEY>
         <DTTM>date, time e.g 2005-01-03 23:57:21</DTTM>
         <TYPE>1=access granted, 2=access denied</TYPE>
         <REASON>reason code from table below</REASON>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Type Code and Reason Code tables

The *GetCardAccessDetails*, *GetAccessCardDetails,* and *GetAccessDataLog* commands return type codes and reason codes. These tables describe what these can be. For valid accesses, no reason code is returned:

Type code	Description
1	Valid access (no reason code supplied)
2	Invalid access
37	Elevator valid access (no reason code supplied)
38	Elevator invalid access
64	Access not completed

Reason codes are provided only for invalid access attempts, and are described in the table below:

Reason code	Description
1	Card not in local database
2	Card not in S2NC database
3	Wrong time
4	Wrong location
5	Card misread
6	Tailgate violation
7	Anti-passback violation
8	--unused--
9	Wrong day
10 – 13	--unused--
14	Card expired
15	Card bit length mismatch
16	Wrong Day
17	Threat Level (prevented access)

By way of example, a call such as:

```
<NETBOX-API>
   <COMMAND name=”GetAccessDataLog” num=”1”>
      <PARAMS>
         <LOGID>999</LOGID>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

might return a result such as:

```
<NETBOX>
   <RESPONSE command=”GetAccessDataLog” num=”1”>
      <CODE>999</CODE>
      <DETAILS>
         <PERSONID>1000</PERSONID>
         <READER>Garage-in</READER>
         <READERKEY>5</READERKEY>
         <DTTM>2005-01-03 23:57:21</DTTM>
         <TYPE>2</TYPE>
         <REASON>4</REASON>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

In this case, an invalid access attempt at 23:57:21 on January 3, 2005 occurred at the “Garage-in” reader by the Person whose ID is 1000. The reason for rejection was that the card was used at the wrong location.

Calling *EditThreatLevelGroup*

EditThreatLevelGroup is called to add (if the group does not exist) or edit (if it does) a threat level group. A call might appear like:

```
<NETBOX-API>
   <COMMAND name=”EditThreatLevelGroup” num=”1”>
      <PARAMS>
         <LEVELGROUPNAME>Hot</LEVELGROUPNAME>
         <LEVELNAMES>
            <LEVELNAME>RedLevel</LEVELNAME>
            <LEVELNAME>OrangeLevel</LEVELNAME>
         </LEVELNAMES>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Note that the names in the <LEVELNAMES> are names of individual levels that must already exist in the system.

Calling *GetPortals*

An example of calling *GetPortals* successively if <NEXTKEY> was returned on a previous call is:

```
<NETBOX-API>
   <COMMAND name=”GetPortals” num=”1”>
      <PARAMS>
         <STARTFROMKEY>32</STARTFROMKEY>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Response to the *GetPortals*

Responses to the *GetPortals* command list the portals and their associated card readers:

```
<NETBOX>
   <RESPONSE command=”GetPortals” num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <PORTALS>
            <PORTAL>
               <NAME>name of portal</NAME>
               <PORTALKEY>unique id for portal</PORTALKEY>
               <READERS>
                  <READER>
                     <READERKEY>unique id for reader</READERKEY>
                     <NAME>name of reader</NAME>
                     <PORTALORDER>1 or 2</PORTALORDER>
                  </READER>
                  <READER>
                  …
                  </READER>
               </READERS>
               </PORTAL>
               <PORTAL>
               …
               </PORTAL>
            </PORTALS>
         <NEXTKEY>key of next portal</NEXTKEY>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

The <PORTALORDER> element of a READER indicates whether it is in the first or the second reader position in a Portal. While this is not guaranteed, the first position is always used as the incoming reader; the second position is optional, and is used as the outgoing reader in a dual reader door when it exists.

Calls and Responses to *GetAccessHistory*

GetAccessHistory serves multiple purposes:

- to retrieve at any one point in time a history of accesses that are available on the NetBox;
- over time to retrieve any new accesses that may have occurred.
- To retrieve a longer history of accesses for a particular card.

- ACCESSES – a list of access data log records responsive to the request. Each data log returned is enclosed within <ACCESS>…</ACCESS> tags, and may include the following fields:
    - DTTM – The controller date and time associated with the data log.
    - TYPE – Valid or invalid access. See the table on page 35 for details on the type codes that can be returned.
    - PORTALKEY – unique identifier for the portal that matches to the GetPortals query.
    - READERKEY – unique identifier for the reader that had the access

These three examples are illustrated below.

```
<NETBOX-API>
   <COMMAND name=”GetAccessHistory” num=”1” dateformat=”tzoffset”>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

This will return the most recent access history records, in reverse time order, up to some internal maximum, for example:

```
<NETBOX>
   <RESPONSE command=” GetAccessHistory” num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <ACCESSES>
            <ACCESS>
               <LOGID>402</LOGID>
               <PERSONID>uid45</PERSONID>
               <READER>reader 1</READER>
               <DTTM>2006-06-23 10:31:06 -0400</DTTM>
               <TYPE>1</TYPE>
               <READERKEY>3</READERKEY>
               <PORTALKEY>30</PORTALKEY>
            </ACCESS>
            <ACCESS>
               <LOGID>397</LOGID>
               <PERSONID>uid23</PERSONID>
               <READER>reader 2</READER>
               <DTTM>2006-06-23 10:15:06 -0400</DTTM>
               <TYPE>1</TYPE>
               <READERKEY>5</READERKEY>
               <PORTALKEY>15</PORTALKEY>
            </ACCESS>
            …
         </ACCESSES>
         <NEXTLOGID>310</NEXTLOGID>
      </DETAILS>
```

```
   </RESPONSE>
</NETBOX>
```

A subsequent call using the <NEXTLOGID> value will retrieve the next older chunk of log records, e.g.:

```
<NETBOX-API>
   <COMMAND name=”GetAccessHistory” num=”1” dateformat=”tzoffset”>
      <PARAMS>
        <STARTLOGID>310</STARTLOGID>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Another call later using <AFTERLOGID> will retrieve any records which have been created since the first call, e.g.:

```
<NETBOX-API>
   <COMMAND name=”GetAccessHistory” num=”1” dateformat=”tzoffset”>
      <PARAMS>
        <AFTERLOGID>402</AFTERLOGID>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

These would be returned in ascending order, going forwards, e.g.:

```
<NETBOX>
   <RESPONSE command=” GetAccessHistory” num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <ACCESSES>
            <ACCESS>
               <LOGID>405</LOGID>
               <PERSONID>uid4</PERSONID>
               <READER>reader 2</READER>
               <DTTM>2006-06-23 10:31:06 -0400</DTTM>
               <TYPE>1</TYPE>
               <READERKEY>5</READERKEY>
               <PORTALKEY>30</PORTALKEY>
            </ACCESS>
         </ACCESSES>
         <NEXTLOGID>-1</NEXTLOGID>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

This indicates that only 1 new entry was found. Let's say after this another call is made, e.g.:

```
<NETBOX-API>
   <COMMAND name=”GetAccessHistory” num=”1” dateformat=”tzoffset”>
      <PARAMS>
         <AFTERLOGID>405</AFTERLOGID>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

If there are no more, it will return “NOT FOUND”, i.e.:

```
<NETBOX>
   <RESPONSE command=” GetAccessHistory” num=”1”>
      <CODE>NOT FOUND</CODE>
   </RESPONSE>
</NETBOX>
```

This next query is for entries matching a particular access card. It is intended as a following call to the *GetCardAccessDetails* and *GetAccessCardDetails* calls.

```
<NETBOX-API>
   <COMMAND name=”GetAccessHistory” num=”1” dateformat=”tzoffset”>
      <PARAMS>
         <ENCODEDNUM>3527</ENCODEDNUM>
         <CARDFORMAT>Code30</CARDFORMAT>
         <STARTLOGID>310</STARTLOGID>
         <MAXRECORDS>100</MAXRECORDS>
         <OLDESTDTTM>2006-06-22 00:00:01</OLDESTDTTM>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Notes on TimeSpecs and TimeSpecGroups

TimeSpecs and TimeSpecGroups are tightly related. The rest of the S2 system refers to time schedules using only TimeSpecGroups. A TimeSpecGroup may be constructed out of one or more TimeSpecs. This is the only use in version 2.5 and later for TimeSpecs.

The S2 system is initialized with 2 TimeSpecs and 2 corresponding TimeSpecGroups: “Always” and “Never.” The “Always” and “Never” Time Specs are returned in a GetTimeSpecs call, and can be used in constructing a TimeSpecGroup.

Likewise, the “Always” and “Never” Time Spec Groups are returned in a GetTimeSpecGroups call.

For *every* TimeSpec that is created, either through the UI or through the NBAPI, a corresponding TimeSpecGroup is automatically created. This TimeSpecGroup is the “singular” TimeSpecGroup that only contains the TimeSpec. Likewise, whenever a TimeSpec is modified or deleted, a corresponding TimeSpecGroup is modified or deleted.

As with the system-initialized TimeSpecGroups, the “singular” TimeSpecGroups constructed by adding a new TimeSpec are returned in a call to “GetTimeSpecGroups.”

While you can construct a new TimeSpecGroup out of any combination of existing TimeSpecs, you cannot modify or delete any of the “singular” TimeSpecGroups that either existed at system

initialization time, or were created by adding new Time Specs. Nor can you modify or delete the "Always" and "Never" Time Specs that exist at system initialization time.

Calling *GetTimeSpec*

This works by supplying a TimeSpec Key. This will typically come from the list of Time Spec Keys in a Time Spec Group.

```
<NETBOX-API>
   <COMMAND name="GetTimeSpec" num="1">
      <PARAMS>
         <TIMESPECKEY>3</TIMESPECKEY>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Response to *GetTimeSpec*

Response to the *GetTimeSpec* command:

```
<NETBOX>
   <RESPONSE command="GetTimeSpec" num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <TIMESPEC>
            <TIMESPECKEY>unique id for timespec</TIMESPECKEY>
            <NAME>name of timespec</NAME>
            <DESCRIPTION>description for timespec</DESCRIPTION>
            <MONDAY>TRUE</MONDAY>
            <TUESDAY>TRUE</TUESDAY>
            …
            <STARTTIME>HH:MM</STARTTIME>
            <ENDTIME>HH:MM</ENDTIME>
            <HOLIDAYGROUPS>1,2,3</HOLIDAYGROUPS>
         </TIMESPEC>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Calling *GetTimeSpecs*

This works identically to *GetPortals*, in that a “STARTFROMKEY” can be passed in to return the next set of Time Specs, if more than one call needs to be made.

Response to *GetTimeSpecs*

Responses to the *GetTimeSpecs* command list time specs

```
<NETBOX>
   <RESPONSE command=”GetTimeSpecs” num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <TIMESPECS>
            <TIMESPEC>
               <TIMESPECKEY>unique id for timespec</TIMESPECKEY>
               <NAME>name of timespec</NAME>
               <DESCRIPTION>description for timespec</DESCRIPTION>
               <MONDAY>TRUE</MONDAY>
               <TUESDAY>TRUE</TUESDAY>
               …
               <STARTTIME>HH:MM</STARTTIME>
               <ENDTIME>HH:MM</ENDTIME>
               <HOLIDAYGROUPS>1,2,3</HOLIDAYGROUPS>
            </TIMESPEC>
            <TIMESPEC>
               …
            </TIMESPEC>
         </TIMESPECS>
         <NEXTKEY>key of next timespec</NEXTKEY>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Calling *GetTimeSpecGroup*

As with *GetTimeSpec*, this specifies a key that might be retrieved from another part of the system.

```
<NETBOX-API>
   <COMMAND name=”GetTimeSpecGroup” num=”1”>
      <PARAMS>
         <TIMESPECGROUPKEY>4</TIMESPECGROUPKEY>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Response to *GetTimeSpecGroup*

Responses to the *GetTimeSpecGroup* command:

```
<NETBOX>
   <RESPONSE command=”GetTimeSpecGroups” num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <TIMESPECGROUP>
            <TIMESPECGROUPKEY>unique id for group</TIMESPECGROUPKEY>
            <NAME>name of timespec</NAME>
            <DESCRIPTION>description for group</DESCRIPTION>
            <TIMESPECKEYS>
               <TIMESPECKEY>ID for timespec</TIMESPECKEY>
               <TIMESPECKEY>ID for timespec</TIMESPECKEY>
               …
            </TIMESPECKEYS>
         </TIMESPECGROUP>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Calling *GetTimeSpecGroups*

This also works identically to *GetPortals*, in that a “STARTFROMKEY” can be passed in to return the next set of Time Spec Groups, if more than one call needs to be made.

Response to *GetTimeSpecGroups*

Responses to the *GetTimeSpecGroups* command list time specs

```
<NETBOX>
   <RESPONSE command=”GetTimeSpecGroups” num=command-number>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <TIMESPECGROUPS>
            <TIMESPECGROUP>
               <TIMESPECGROUPKEY>unique id for group</TIMESPECGROUPKEY>
               <NAME>name of timespec</NAME>
               <DESCRIPTION>description for group</DESCRIPTION>
               <TIMESPECKEYS>
                  <TIMESPECKEY>ID for timespec</TIMESPECKEY>
                  <TIMESPECKEY>ID for timespec</TIMESPECKEY>
                  …
               </TIMESPECKEYS>
            </TIMESPECGROUP>
            <TIMESPECGROUP>
               …
            </TIMESPECGROUP>
         </TIMESPECGROUPS>
         <NEXTKEY>key of next group</NEXTKEY>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Errors in API Processing

Errors in processing of specific, well formed API commands are explained above. Errors which prevent the processing of commands return a special XML blob:

```
<NETBOX>
   <RESPONSE>
      <APIERROR>apiErrorCode</APIERROR>
   </RESPONSE>
</NETBOX>
```

Where *apiErrorCode* is defined as in the table below:

apiError	Code	Description
API_INIT_FAIL	1	Database error (database not running, etc.)
API_DISABLED	2	The API processor is not activated for this S2 NetBox
API_NOCOMMAND	3	No APIcommand parameter was passed to the API processor
API_PARSE_ERROR	4	The APIcommand data could not be parsed by the XML parser
API_AUTH_FAILURE	5	API authorization failure (if authorization enabled in the user interface)
API_UNKNOWN_COMMAND	6	The API processor did not recognize the command passed in the APIcommand parameter.

Message Authentication Code

The *message authentication code* (MAC) is a code based on the SHA-1 hashing algorithm that is designed to be unique for every message and impossible to reverse in practice. The MAC is transmitted in the <MAC> tag of the message and has the form:

RAN1 (1-5)	RAN2 (6-10)	SEQ # (11-20)	SHA-1 result as 40 hexadecimal digits (21-60)

where RAN1 and RAN2 are two random numbers, each of five decimal digits; SEQ # is the zero-padded 10-digit sequence number that is incremented for each message transmitted by the caller; and, the balance is the 40-hex digit SHA-1 hash as produced by the algorithm provided.

Generating the MAC

Code in C is provided by S2 and other organizations to calculate the SHA-1 digest used in the MAC. S2 also provides a utility function, *generate_mac*, to create the MAC from caller-provided inputs as follows:

```
#define SHA_SECRET_SZ 8             //the secret is 8 bytes long

// generate_mac:
// takes the msg, seq and secret and generates a SHA_MAC_LEN + 1
// byte mac (null terminated) which is then stored in mac
// and a pointer to mac is returned
```

```
//
// NOTES:
//
//   1. msg must be null terminated and should not contain the
//   <MAC>...</MAC> tags.
//
//   2. secret must be null terminated and < SHA_SECRET_SZ bytes long
//
unsigned char *generate_mac(unsigned char *msg, unsigned long seq,
unsigned char *secret, unsigned short rand1, unsigned short rand2,
unsigned char *mac);
```

Examples

For example, the XML generated to add a person might appear as follows:

```
<NETBOX-API>
   <COMMAND name=”EditPerson” num=”1”>
      <PARAMS>
         <LASTNAME>Doe</LASTNAME>
         <FIRSTNAME>John</FIRSTNAME>
         <ACCESSLEVELS>
            <ACCESSLEVEL>GeneralAccess</ACCESSLEVEL>
            <ACCESSLEVEL>ParkingAccess</ACCESSLEVEL>
         </ACCESSLEVELS>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

The command would return information such as:

```
<NETBOX>
   <RESPONSE command=”EditPerson” num=”1”>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <PERSONID>_123</PERSONID>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

If an error had occurred, then the ERRMSG tag would give a text description of the error condition as in:

```
<NETBOX>
   <RESPONSE command=”EditPerson” num=”1”>
      <CODE>FAIL</CODE>
      <DETAILS>
         <ERRMSG>
            Access level does not exist “ParkingAccess”
         </ERRMSG>
      </DETAILS>
   </RESPONSE>
</NETBOX>
```

Once the person record has been instantiated, a credential (card) can be added to it as follows:

```
<NETBOX-API>
   <COMMAND name=”AddCredential” num=”1”>
      <PARAMS>
         <PERSONID>_123</PERSONID>
         <CARDNUM>AF2621B5</CARDNUM>
         <CARDFORMAT>Wiegand26</CARDFORMAT>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Because multiple commands can be included in a single XML call, the creation of a Person and association of credentials can be accomplished in one transaction (if the caller provides the PERSONID):

```
<NETBOX-API>
   <COMMAND name=”EditPerson” num=”1”>
      <PARAMS>
         <LASTNAME>Doe</LASTNAME>
         <FIRSTNAME>John</FIRSTNAME>
         <ACCESSLEVELS>
            <ACCESSLEVEL>GeneralAccess</ACCESSLEVEL>
            <ACCESSLEVEL>ParkingAccess</ACCESSLEVEL>
         </ACCESSLEVELS>
         <PERSONID>X1000</PERSONID>
      </PARAMS>
   </COMMAND>
   <COMMAND name=”AddCredential” num=”2”>
      <PARAMS>
         <PERSONID>X1000</PERSONID>
         <CARDNUM>AF2621B5</CARDNUM>
         <CARDFORMAT>Wiegand26</CARDFORMAT>
      </PARAMS>
   </COMMAND>
   <MAC>  …authentication code…  </MAC>
</NETBOX-API>
```

Assuming that the commands above were successfully executed by the S2 NetBox, the response would be as follows:

```
<NETBOX>
   <RESPONSE command=”EditPerson” num=”1”>
      <CODE>SUCCESS</CODE>
      <DETAILS>
         <PERSONID>X1000</PERSONID>
      </DETAILS>
   </RESPONSE>
   <RESPONSE command=”AddCredential” num=”2”>
      <CODE>SUCCESS</CODE>
   </RESPONSE>
</NETBOX>
```