

PASSPORT READER NETAPI INTERFACE REFERENCE MANUAL

Document version: 2023-08-01

Table of Contents

| | |
|---|----|
| THE PR NETAPI..... | 3 |
| 1. Setup on the Osmond N Device web interface | 3 |
| 2. Setup server on PC..... | 4 |
| 3. Setup Client..... | 4 |
| THE FUNCTIONS OF THE NETAPI..... | 5 |
| 1. The ResponseHandler | 5 |
| 2. Namespaces..... | 6 |
| 3. Connection..... | 7 |
| 3.1. HTTP/HTTPS Login | 7 |
| 3.2. HTTP/HTTPS Logout | 8 |
| 3.3. WebSocket Connection..... | 8 |
| 3.4. WS/WSS Login..... | 8 |
| 3.5. WebSocket Disconnection..... | 9 |
| 3.6. Setup Proxy | 9 |
| 3.7. List of the Device Servers | 9 |
| 3.8. SDK Version Query | 10 |
| 3.9. Request for the Device List..... | 10 |
| 3.10. Query of the Enumerated Values of the SDK | 11 |
| 3.11. Query of the SDK Functions | 11 |
| 4. Using the Device | 12 |
| 4.1. Gain Access to the Device | 12 |

| | | |
|-------|--|----|
| 4.2. | Setting of an Operating Parameter | 12 |
| 4.3. | Query of an Operating Parameter | 13 |
| 4.4. | Subscribe to Event Notification | 13 |
| 4.5. | Generating an Event..... | 13 |
| 4.6. | Unsubscribe from Event Notification | 14 |
| 4.7. | Finish Using the Device..... | 14 |
| 5. | Imaging..... | 15 |
| 5.1. | Query of the Information of the Image Scanner..... | 15 |
| 5.2. | Scanning | 15 |
| 5.3. | Deletion of All the Images and Document Data Stored in Memory | 16 |
| 5.4. | Deletion of the Images and Document Data of the Last Scanned Document Page | 16 |
| 5.5. | Image Query | 17 |
| 5.6. | Query of Image Information | 18 |
| 5.7. | Launch of the Background Tasks of the Imaging System | 18 |
| 5.8. | Finishing the Background Tasks of the Imaging System | 19 |
| 6. | Optical Reading | 20 |
| 6.1. | Engine Information Query | 20 |
| 6.2. | Query of the Information of the Available Licenses | 20 |
| 6.3. | Query of the Installed Engines | 20 |
| 6.4. | Recognition of a Scanned Page | 21 |
| 6.5. | Query of One or More Recognized Data Field Image | 22 |
| 6.6. | Summary of Two Data Description Structure..... | 22 |
| 6.7. | Query of the Whole Data Description Structure..... | 23 |
| 7. | Electronic Card Reading | 24 |
| 7.1. | Query of the Electronic Card Readers | 24 |
| 7.2. | Query of the Information of the Card Reader..... | 24 |
| 7.3. | Search for the Electronic Card | 24 |
| 7.4. | Opening the Electronic Card for Use | 25 |
| 7.5. | Query of the Card Information | 25 |
| 7.6. | Launch of the Card Reading in the Background..... | 26 |
| 7.7. | Query of the Next Recommended Card Authentication | 26 |
| 7.8. | Performing Card Authentication..... | 27 |
| 7.9. | Query of the Card Applications | 27 |
| 7.10. | Query of the Card Files..... | 28 |
| 7.11. | Query of a File | 28 |
| 7.12. | Checking the File Integrity | 29 |
| 7.13. | Conversion of the File Identifier..... | 29 |
| 7.14. | Summary of the Card Authentication Results | 29 |
| 7.15. | Disconnect the Card | 30 |
| 8. | Admin Functions..... | 31 |
| 8.1. | Query of the Usage Information | 31 |
| 8.2. | Sending Message to User..... | 32 |
| 8.3. | Forced Logout of a User | 32 |
| 9. | JavaScript Task Generating Functions | 33 |
| | CONTACT INFORMATION..... | 34 |



THE PR NETAPI

The PR NetAPI is the network version of the Passport Reader SDK. Its interface implements WebSocket communication with JSON-RPC format packages. This WebSocket channel is either provided by an Osmond N network device or by the NetAPI service running on PC.

The NetAPI is designed to control remote Osmond N devices via Ethernet connection as well as Windows/Linux connected legacy USB document scanners from not natively supported operating systems.

Additional uses:

- Supports running UWP programs on Windows via localhost connection
- It helps to optimize memory usage by balancing load between client and server
- The Passport Reader software package includes a NetAPI client that allows accessing all supported document reader devices through the conventional SDK as well.
- The standalone version of the .NET interface can be operated without the installation of the PR system as well.

1. SETUP ON THE OSMOND N DEVICE WEB INTERFACE

1. Create a user with 'NAI user' role. Only one user can be logged in at the same time.
2. Upload a HTTPS certificate. NetAPI operates via HTTPS communication only.
3. Set the operating mode of the device to 'NAI' mode.
4. The NetAPI is accessible via the same port number as the web interface.

2. SETUP SERVER ON PC

1. Create a NetAPI user with the PRDTool program. The user needs admin or user role. Maximum 5 users can be logged in at the same time in order to use several connected devices. The user sessions can be managed with admin role.
2. The operation parameters can be set with the PRDTool program:
 - Port number (default: 8000)
 - SSL certificate file and SSL private key file for encrypted communication
If the encrypted communication is configured, the server cannot be accessed without encryption.
 - Enable external access
If enabled, the server accepts requests from other devices. Otherwise, communication is restricted to localhost.
 - RFID certificate folder
The path comprising files required for Passive and Terminal Authentications
3. The NetAPI service is realized by the prwebsrv program that can operate as a Windows service or Linux daemon. The server can be turned on/off with the PRDTool program as well.
On Windows, the service state can be queried from command line with the ``prwebsrv --svc-query`` command. If the program is executed in foreground with the ``prwebsrv --showlog`` command, it displays the communication packages to assist developer.
4. The configuration files - prwebsrv.json and the webusr.json - can be copied freely between computers. Uninstalling the Passport Reader software package removes these files.

3. SETUP CLIENT

The NetAPI client is part of the Passport Reader software. In order to use it, set the following properties within the default/pr node in the gxsd.dat file manually, or by your client program:

- ipdev/url – Server IP address (or domain name) and port number.
- ipdev/user – Username.
- ipdev/password – Password. Not recommended, but possible to set it in the gxsd.dat file.
- ocr_module - OCR tasks can be performed on client side or on server side. Set this property to ``procr-ip`` to perform OCR on server side. If the server and the client are on the same PC (localhost connection), do not apply this setting in the gxsd.dat file.

THE FUNCTIONS OF THE NETAPI

In the following the [JSON-RPC 2.0](#) commands that can be called through WebSocket and the commands of the JavaScript implementation that can be found in the SDK will be explained.

1. THE RESPONSEHANDLER

The functions of the js interface generally contain a responseHandler parameter. This is the handler function of response to requests. The following types can be specified:

- ok_handler_func
- [common_handler_func]
- [ok_handler_func, failure_handler_func]

Null function can also be defined instead of whichever function. In case of common_handler_func, the success of the call can be determined from its parameter value.

2. NAMESPACES

In order to avoid name collision, the js interface uses namespace classes. The initial global variable is the **Pr22**. The resulting structure is the following:

- Pr22.**DocumentReader**: DeviceManager Class
The returning value is hereinafter referred to as **pr**.
 - pr.**getScanner**: ImageScanner Class
The returning value is hereinafter referred to as **scanner**.
 - pr.**getEngine**: OcrEngine Class
The returning value is hereinafter referred to as **engine**.
 - pr.**getCardReader**(index) ElectronicCardReader Class
The returning value is hereinafter referred to as **reader**.
 - reader.**getCard**(connectResponse) ElectronicCard Class
The returning value is hereinafter referred to as **card**.
- Pr22.**DocScannerTask**: For imaging tasks
- Pr22.**FreerunTask**: For background tasks of the imaging system
- Pr22.**EngineTask**: For image processing tasks
- Pr22.**ECardTask**: For reading tasks of the electronic card

3. CONNECTION

In order to establish connection with Osmond N, perform the following steps:

1. Sign in with the HTTPS login command
2. Open WSS connection

On PC server, there are two ways:

- With HTTP/HTTPS protocol:
 1. Sign in with the HTTP/HTTPS login command
 2. Open WS/WSS connection
- Without HTTP/HTTPS protocol:
 1. Open the WS/WSS connection
 2. Call the WS/WSS login function

The user session has a time limit. The session timeout value is reset by every message. The server sends notifications 60 and 30 seconds before session expiry, through **User Message** event.

3.1. HTTP/HTTPS LOGIN

Send **POST** `/netapi/login` request to the server with the Username and Password fields located in the header. Json structure returned in message body of the response contains the `api_token` element. This is to be handed over in the Token header field when opening WebSocket.

parameter:

- Username: Username
- Password: Password

response:

- `api_token`: Data to be forwarded for opening WebSocket or http/https logout

parameter:

- Token: Value of the `api_token` is to be forwarded upon opening ws/wss

3.2. HTTP/HTTPS LOGOUT

Send **GET** `/netapi/logout` request to the server with the Token field located in the header.

parameter:

- Token: Value of the api_token is to be forwarded in order to log out

3.3. WEBSOCKET CONNECTION

When using HTTP/HTTPS login:

- Establish connection to the following endpoint: ``wss://hostname:port/netapi/control``
- Send the Token: api_token as header field.

WS -

```
js Pr22.connect(host, onConnectFinishedHandler, onConnectFailedHandler, onConnectionClosedHandler)
```

parameter:

- host: If you are using our proxy (see [setProxy\(\)](#) for more information), then this value must be the name of a device server (returned by listServers() method). Otherwise, the value must be the URL of a device server including the protocol (http/https) and the port number.
- onConnectFinishedHandler: Callback function in case of successful connection.
- onConnectFailedHandler: Callback function in case of unsuccessful connection.
- onConnectionClosedHandler: Callback function to be called in case of closing the WebSocket communication

3.4. WS/WSS LOGIN

```
ws Login User, Password
js Pr22.login (user, passw, responseHandler)
```

parameter:

- User: Username
- Password: Password

3.5. WEBSOCKET DISCONNECTION

This function closes the WebSocket channel and logs out from the server.

```
ws -  
js Pr22.disconnect()
```

3.6. SETUP PROXY

Osmond N devices block direct connection from JavaScript due to cross-site scripting safety. A sample proxy server is annexed to the js interface as a solution. This sample proxy server can store a list of device servers and the required user name/password pairs (without encryption). When the proxy is set in the client program with the setProxy() method, then:

- all the communication will be transmitted through the proxy server,
- the listServers() method can get the list of registered device servers,
- the connect() method automatically executes the necessary http/https login method.

```
ws -  
js Pr22.setProxy(host)
```

parameter:

- host: URL of the proxy excluding protocol and including port number

3.7. LIST OF THE DEVICE SERVERS

This method returns the name-list of the registered device servers in case of using proxy (see [setProxy\(\)](#) for more information). The entries can be used in the connect() method.

```
ws -  
js Pr22.listServers(responseHandler)
```

3.8. SDK VERSION QUERY

```
ws GetApiVersion           Interfaceoptional  
js Pr22.getVersion        (responseHandler)
```

parameter:

- Interface: It should be handed over the version number of the js interface used in development in order to avoid possible future compatibility problems.

response:

- Interface: Version of the javascript interface. If it is not handed over as input on the ws interface, then there is no such item.
- WebSDK: Version of the prwebsrv program.
- System: Version of the installed Passport Reader software.

3.9. REQUEST FOR THE DEVICE LIST

```
ws ListDevices  
js Pr22.listDevices(responseHandler)
```

response:

- Array containing device names.

3.10. QUERY OF THE ENUMERATED VALUES OF THE SDK

This function is mostly a hint for developers, so it is not used during operation. It returns the enum types used by SDK when calling it without parameter. By defining an enum type it returns its possible values.

```
ws ListEnumValues EnumType
js Pr22.listEnumValues (enumType, responseHandler)
```

parameter:

- EnumType: Null
- EnumType: Name of the enum type.

response:

- EnumTypes: Array of enum types names.
- Requested enum type name: Array of enum values names.

3.11. QUERY OF THE SDK FUNCTIONS

This function is mostly a hint for developers, so it is not used during operation. It returns the available functions list.

```
ws ListMethods
js Pr22.listMethods(responseHandler)
```

response:

- Array containing functions names.

4. USING THE DEVICE

4.1. GAIN ACCESS TO THE DEVICE

```
ws UseDevice Device <> Index, Modeoptional
js pr.useDevice (deviceName, responseHandler, mode)
```

parameter:

- Device, deviceName as string: Device name.
- Index, deviceName as number: Serial number of the device in the device list.
- Mode: RFU

response:

- true

4.2. SETTING OF AN OPERATING PARAMETER

```
ws SetProperty ["prop", "value"]
js pr.setProperty (prop, value, responseHandler)
```

parameter:

- prop: Property to be set up.
- value: New value of the property.

response:

- true

4.3. QUERY OF AN OPERATING PARAMETER

```
ws GetProperty "prop"  
js pr.getProperty (prop, responseHandler)
```

parameter:

- prop: Property to be read.

response:

- The value of the property.

4.4. SUBSCRIBE TO EVENT NOTIFICATION

Json rpc notification is received concerning the monitored event.

```
ws SetEvent EventType  
js pr.setEventHandler (eventType, eventHandler, responseHandler)
```

parameter:

- EventType: Event which has to be monitored.
- eventHandler: Javascript event handler routine.

response:

- true

4.5. GENERATING AN EVENT

```
ws TriggerEvent EventType  
js pr.triggerEvent (eventType, responseHandler)
```

parameter:

- EventType: Event which has to be generated.

response:

- true

4.6. UNSUBSCRIBE FROM EVENT NOTIFICATION

Events can still arrive from the waiting queue after calling of this method is finished.

```
ws ClearEvent                               EventType
js pr.clearEventHandler (eventType, eventHandler, responseHandler)
```

parameter:

- EventType: Event which has to be monitored.
- eventHandler: Javascript event handler routine.

response:

- true

4.7. FINISH USING THE DEVICE

```
ws CloseDevice
js pr.closeDevice (responseHandler)
```

response:

- true

5. IMAGING

5.1. QUERY OF THE INFORMATION OF THE IMAGE SCANNER

WS **GetScannerInfo**

```
js scanner.getInfo(responseHandler)
```

response:

- The automatic json conversion of the passport reader data description structure. It contains the list of the available lights and window objects as well as the version numbers.

5.2. SCANNING

WS **Scan**

Lights, Window_{optional}, PagePos_{optional}

```
js scanner.scan
```

```
(task, responseHandler, pagePosition)
```

parameter:

- Lights: Array of the lights to be prepared.
- task: Light identification string, or array of several light strings, or Pr22.DocScannerTask.
- Window: Ordinal number of the window object. There is only one on most devices.
- pagePosition: Logical page on which images should be stored in the memory.

response:

- true

The scanning is asynchronous which means the response is immediately received, even if the process launch fails. Therefore, one should wait for the special events.

Available events during scanning: **ScanStarted**, **ImageScanned**, **ScanFinished**, **DocFrameFound**

"All" can be defined at lights followed by the exceptions with the "-" prefix.

5.3. DELETION OF ALL THE IMAGES AND DOCUMENT DATA STORED IN MEMORY

```
ws CleanUp                                LastPage = false  
js scanner.cleanUpData                    (responseHandler)
```

response:

- true

5.4. DELETION OF THE IMAGES AND DOCUMENT DATA OF THE LAST SCANNED DOCUMENT PAGE

This function should be called if an incorrect page has been read during multiple-page document scanning.

```
ws CleanUp                                LastPage = true  
js scanner.cleanUpLastPage                (responseHandler)
```

response:

- true

5.5. IMAGE QUERY

ws **GetImage** Light, Page_{optional}, DocView_{optional}, ...
 js scanner.**getImage** (selection, responseHandler, options)

parameter:

- Light: Light of the queried image.
- Page: Page number of the queried image.
- DocView: 1, if the cropped and rotated document image is required,
2, if the cropped and rotated document image is preferred, but the image of the reader is also acceptable.
- selection: Light identification string, or an array in which the page number as a numerical value and the "DocView" or "PreferDocView" as a text are optionally included besides the light identification.
- options: In case of js interface the "options" is an object, in case of ws interface the options are on the same level as the other parameters. These are:
 - Dpi: Required image resolution in dpi.
 - Format: Mime type or value of the ImageFileFormat enum.
 - Compression parameter depending on format: Quality(Jpeg), Rate(Jpeg 2000), BitRate(Wsq)
 - Comment: Comment to be written to image file, if the format allows.
 - Base64: True/false. Base64 embedded into Json or binary data transmission.

response:

- Image: Base64 format image, if it has been requested this way.
- Mime: Mime type in case of using Base64, if exists.

5.6. QUERY OF IMAGE INFORMATION

```
ws GetImageInfo Light, Pageoptional, DocViewoptional  
js scanner.getImageInfo (selection, responseHandler)
```

parameter:

- Light: Light of the queried image.
- Page: Page number of the queried image.
- DocView: 1, if the cropped and rotated document image is required,
2, if the cropped and rotated document image is preferred, but the image of the reader is also acceptable.
- selection: Light identification string, or an array in which the page number as a numerical value and the "DocView" or "PreferDocView" as a text are optionally included besides the light identification.

response:

- The automatic json conversion of the passport reader data description structure, supplemented by Dpi and Size information.

5.7. LAUNCH OF THE BACKGROUND TASKS OF THE IMAGING SYSTEM

```
ws StartTask Task  
js scanner.startTask (task, responseHandler)
```

parameter:

- Task: String or Pr22.**FreerunTask**.

response:

- true

For proper operation one should subscribe to the corresponding event too.

5.8. FINISHING THE BACKGROUND TASKS OF THE IMAGING SYSTEM

```
ws StopTask Task
js scanner.stopTask (task, responseHandler)
```

parameter:

- Task: String or Pr22.**FreerunTask**.

response:

- true

6. OPTICAL READING

6.1. ENGINE INFORMATION QUERY

ws **GetEngineInfo**

```
js engine.getInfo(responseHandler)
```

response:

- The automatic json conversion of the passport reader data description structure. It contains the engine version and the license information.

6.2. QUERY OF THE INFORMATION OF THE AVAILABLE LICENSES

ws **ListLicenses**

options

```
js engine.listLicenses(responseHandler, options)
```

parameter:

- options: Object. If it contains Raw = true, numeric license identifiers are displayed in the response.

response:

- Array of objects, which contains the license type and its expiry date.

6.3. QUERY OF THE INSTALLED ENGINES

ws **ListEngines**

```
js engine.listEngines(responseHandler)
```

response:

- Array containing engines names.

6.4. RECOGNITION OF A SCANNED PAGE

```
ws Analyze PageOptional, Task, LightsOptional  
js engine.analyze (page, task, responseHandler, imgFilter)
```

parameter:

- Page: Ordinal number of the page to be read.
- Task: Array of the field identifiers to be queried or Pr22.**EngineTask**.
- Lights: Light filter array.
- imgFilter: Light identification string or array of lights, or Pr22.**DocScannerTask**.

response:

- The automatic json conversion of the passport reader data description structure.

The "source" and "fieldid" elements must be separated by dots at the field identifiers.

"**All**" can be defined at tasks followed by the exceptions with the "-" prefix.

Unlike PC SDK, the "**All**" does not contain the "**LowLevel**" and "**FormatCheck**" IDs. By default, the Field frames are not returned. Those can be retrieved by the mentioned two IDs or by the "**Frames**" ID.

"**All**" can be defined at light filters followed by the exceptions with the "-" prefix.

6.5. QUERY OF ONE OR MORE RECOGNIZED DATA FIELD IMAGE

```
ws GetFieldImage          Doc, Field, ...
js engine.getFieldImage   (doc, field, responseHandler, options)
```

parameter:

- Doc: Identifier of the data description structure received beforehand:
Data member named Apid
- Field: Field identifier or array of identifiers of the required image. The "source" and "fieldid" elements must be separated by dots
- options: In case of js interface the "options" is an object, in case of ws interface the options are on the same level as the other parameters. These are:
 - Dpi: Required image resolution in dpi.
 - Format: Mime type or value of the ImageFileFormat enum.
 - Compression parameter depending on format: Quality(Jpeg), Rate(Jpeg 2000), BitRate(Wsq)
 - Comment: Comment to be written to image file, if the format allows.
 - Base64: True/false. Base64 embedded into Json or binary data transmission.

response:

- Image: Base64 format image, if it has been requested this way.
- Mime: Mime type in case of using Base64, if exists.
- In case of array range query, the Base64 result is also array range.

6.6. SUMMARY OF TWO DATA DESCRIPTION STRUCTURE

The result contains "Preferred" fields and *FieldCompare* results in case of multiple data.

```
ws MergeDocuments       One, Other
js engine.merge         (one, other, responseHandler)
```

parameter:

- One, Other: Identifier of the data description structure received beforehand:
Data member named Apid

response:

- The automatic json conversion of the summarized data description structure.

6.7. QUERY OF THE WHOLE DATA DESCRIPTION STRUCTURE

```
ws GetRootDocument                Format, ...
js engine.getRootDocument          (format, responseHandler, options)
```

parameter:

- **Format:** Mime type or enum value of the DocFileFormat. Null for the raw json structure.
- **Options:** In case of js interface the "options" is an object, in case of ws interface the options are on the same level as the other parameters. These are:
 - **Base64:** True/false. Base64 embedded into Json or binary data transmission. The xml formats are always embedded into Json, even in case of Base64 = false.

response:

- The automatic json conversion of the passport reader data description structure in case of format = null.
- In case of using Base64 or xml type:
 - **RootDoc:** The converted data description structure.
 - **Mime:** Mime type, if exists.
 - **Ext:** Recommended extension.

7. ELECTRONIC CARD READING

7.1. QUERY OF THE ELECTRONIC CARD READERS

```
ws ListCardReaders  
js pr.listCardReaders(responseHandler)
```

response:

- Array containing the names of the readers.

7.2. QUERY OF THE INFORMATION OF THE CARD READER

```
ws GetReaderInfo Reader  
js reader.getInfo (responseHandler)
```

parameter:

- Reader: Name of the card reader.

response:

- The automatic json conversion of the passport reader data description structure. It contains the version numbers.

7.3. SEARCH FOR THE ELECTRONIC CARD

```
ws ListCards Reader  
js reader.listCards (responseHandler)
```

parameter:

- Reader: Name of the card reader.

response:

- Array containing the serial numbers of the cards.

7.4. OPENING THE ELECTRONIC CARD FOR USE

```
ws ConnectCard           Reader, Card
js reader.connectCard    (cardNo, responseHandler)
```

parameter:

- Reader: Name of the card reader.
- Card: The index number of the card in the serial number list.

response:

- CardId: The serial number of the card.

7.5. QUERY OF THE CARD INFORMATION

```
ws GetCardInfo           Card
js card.getInfo           (responseHandler)
```

parameter:

- Card: The serial number of the card.

response:

- The automatic json conversion of the passport reader data description structure. It contains the card type, its parameters and the period of use.

7.6. LAUNCH OF THE CARD READING IN THE BACKGROUND

ws **StartRead** Card, Files, AuthLevel_{optional}, ...
 js card.**startRead** (task, responseHandler, options)

parameter:

- Card: The serial number of the card.
- Task: File identification string, or array of file identifiers, or Pr22.**ECardTask**.
- Files: File identification array.
- AuthLevel: Authentication Level.
- options: In case of js interface the "options" is an object, in case of ws interface the options are on the same level as the other parameters. These are:
 - AutoMrz: In case of PACE/BAC the last read data from the images are used automatically.
 - FileFormat: If set, the read files are automatically integrated to the **ReadFinished** event.

response:

- true

"All" can be defined at files followed by the exceptions with the "-" prefix.

Available events during reading: **AuthBegin**, **AuthWaitForInput**, **AuthFinished**, **ReadBegin**, **ReadFinished**, **FileChecked**

The process can be interrupted by calling the **StopTask** function and handing over "ECard" parameter.

7.7. QUERY OF THE NEXT RECOMMENDED CARD AUTHENTICATION

ws **GetNextAuthentication** Card, ForceNext
 js card.**getNextAuthentication** (forceNext, responseHandler)

parameter:

- Card: The serial number of the card.
- ForceNext: If set, it skips the next one in line and recommends the following one.

response:

- Auth: The recommended authentication.

7.8. PERFORMING CARD AUTHENTICATION

```
ws Authenticate Card, Auth, Dataconditional  
js card.authenticate (auth, data, responseHandler)
```

parameter:

- Card: The serial number of the card.
- Auth: The authentication to be performed.
- Data: The parameter of the authentication.
 - In case of PACE the password type is to be entered before the string element in the following form: "MRZ:" or "CAN:"
 - In case of SelectApp the enum or integer value can be defined.
 - In case of CompleteTA a Base64 encoded string without particular designation is to be defined.

response:

- true

7.9. QUERY OF THE CARD APPLICATIONS

```
ws ListApplications Card  
js card.listApplications (responseHandler)
```

parameter:

- Card: The serial number of the card.

response:

- Array containing the identifiers of the applications.

7.10. QUERY OF THE CARD FILES

```
ws ListFiles Card
js card.listFiles (responseHandler)
```

parameter:

- Card: The serial number of the card.

response:

- Array containing the identifiers of the files.

7.11. QUERY OF A FILE

```
ws GetFile Card, File, Formatoptional
js card.getFile (file, responseHandler, format)
```

parameter:

- Card: The serial number of the card.
- File: Identifier of the file to be queried. Enum or DGx value.
- Format: Format of the result, the enum value of the CardFileFormat.

response:

In case of Base64 format:

- ECardFile: Identifier of the file.
- ECardFileData: The file content in encoded form.

The automatic json conversion of the passport reader data description structure in case of *Document* or *FullDoc* format.

Only the processed data are included in the *Document* format.

The *FullDoc* format contains the processed data and the binary file in encoded form as well.

7.12. CHECKING THE FILE INTEGRITY

```
ws CheckHash Card, File
js card.checkHash (file, responseHandler)
```

parameter:

- Card: The serial number of the card.
- File: The identifier of the file to be checked. Enum or DGx value.

response:

- true/false

7.13. CONVERSION OF THE FILE IDENTIFIER

```
ws ConvertFileId Card, File
js card.convertFileId file, responseHandler)
```

parameter:

- Card: The serial number of the card.
- File: The file identifier to be managed. Enum or DGx value.

response:

- ID received as parameter: transformed ID.

7.14. SUMMARY OF THE CARD AUTHENTICATION RESULTS

```
ws GetAuthResult Card
js card.getAuthResult (responseHandler)
```

parameter:

- Card: The serial number of the card.

response:

- The automatic json conversion of the passport reader data description structure.

7.15. DISCONNECT THE CARD

```
ws Disconnect Card  
js card.disconnect (responseHandler)
```

parameter:

- Card: The serial number of the card.

response:

- true

8. ADMIN FUNCTIONS

The admin functions are only available in PC version. Only one user can be logged in to the Osmond N NetAPI server at the same time.

8.1. QUERY OF THE USAGE INFORMATION

ws **GetUsageInfo**

js admin.**getUsageInfo** (responseHandler)

response:

- Availability: "Full" text
- Devices: Array of the user data which consists of the following (with the exception of "Used" each one is optional):
 - Device: Device name.
 - Used: Username or False
 - Address: IP address and port number of the user
 - Expire: Remaining time of the session
 - Locked: Time of the device entering service

Usage information is not available with user role. Only the Used: true/false data is available with user role.

8.2. SENDING MESSAGE TO USER

```
ws UserMessage      Address, Message
js admin.userMessage (address, message, responseHandler)
```

parameter:

- Address: IP address and port number of the recipient user, used for logging in to the server
- Message: Message text

response:

- true

The message arrives to the recipient in the **UserMessage** event.

8.3. FORCED LOGOUT OF A USER

```
ws ForceLogout      Address
js admin.forceLogout (address, responseHandler)
```

parameter:

- Address: IP address and port number of the user to be logged out, used for logging in to the server

response:

- true

The server sends notification to the client through **UserMessage** event.

9. JAVASCRIPT TASK GENERATING FUNCTIONS

DocScannerTask.**add**(light)

DocScannerTask.**del**(light)

DocScannerTask.**objectWindow**

EngineTask.**add**(field)

EngineTask.**del**(field)

FreerunTask.**Detection**()

ECardTask.**add**(file)

ECardTask.**del**(file)

ECardTask.**authLevel**

CONTACT INFORMATION

Headquarters:

Adaptive Recognition, Hungary Inc.

Alkotás utca 41 HU

1123 Budapest Hungary

Web: adaptiverecognition.com

Service Address:

Adaptive Recognition, Hungary Inc.

Ipari Park HRSZ1113/1 HU

2074 Perbál Hungary

Web: adaptiverecognition.com/support/

Adaptive Recognition Hungary Technical Support System (ATSS) is designed to provide you the fastest and most proficient assistance, so you can quickly get back to business.

Information regarding your hardware, latest software updates and manuals are easily accessible for customers via our [Documents Site \(www.adaptiverecognition.com/doc\)](http://www.adaptiverecognition.com/doc) after a quick registration.

New User

If this is your first online support request, please contact your sales representative to register you in our Support System. More help [here \(www.adaptiverecognition.com/support\)](http://www.adaptiverecognition.com/support)!

Returning User

All registered ATSS customers receive a personal access link via e-mail. If you previously received a confirmation message from ATSS, it contains the embedded link that allows you to securely enter the support site.