SSG FUSION LTD
Putting You In Control of Your Data

Box 870 • Picture Butte, AB • T0K 1V0 403-382-1568 (Darryl) sales@ssgfusion.com www.ssgfusion.com

Symmetry API Reference

Version 1.0

General

Symmetry is an application which replicates the birth date, breed, and Express Verified portions of the CCIA database by downloading new changes every night. Having this information on site makes it so you can always rely on a very quick response to such queries, even if your internet is slow or not working. Symmetry has two APIs which your application can use to communicate with it to retrieve age verification, breed, and Express Verified information. The preferred API, because it is the fastest and most useful in high volume situations, is the TCP API. The other API is the File API. Both APIs use the same request/response strategy. It is possible for Symmetry to use both APIs at the same time.

Each request sent to Symmetry must be exactly what is expected. No extra spaces, etc. are allowed or a parsing error will be returned. (The only exception to this is if the RFID tag has spaces in it.) The request method name comes first followed by a left parenthesis. If the request has parameters, they follow next, each separated by a comma. Then comes a right parenthesis. If no parameters are required, the parentheses must still be present. If there are parameters, the correct number of parameters for that request must be present.

Results will always follow exact formatting rules as well, making it easy to parse the results on your end. Spaces will not exist, except in the case of a message being returned (i.e. an error message). Boolean values will either be True or False. Dates will always be 8 digits in the format YYYYMMDD. Empty dates will be 00000000. An empty breed will be returned as an empty string. Results with multiple parts will be separated by commas.

Internally, the CCIA database sees RFID numbers as 21 digit numbers. Symmetry will attempt to convert the RFID number you send to it into a proper CCIA tag number. This includes converting country codes to their numeric equivalents. If you pass less than 15 digits, Symmetry will assume the country code to be for Canada (124).

If there is a parsing error, or anything else happens which results in an error, an error message is returned for the response. It will always start with “Error: ” (note the space following the colon) followed by a one line description of the error. Error descriptions are meant to be human readble—that is, they are meant to be displayed somewhere, not parsed. Thus, there is no definitive list of possible error descriptions.

Each request must be followed by exactly one end of line delimiter. Results follow this same rule. This is important in both APIs so the receiving application knows when the full request or response has been received. The end of line delimiter used should be the standard internet (or Windows) end of line regardless of the platform being run on. This is an ASCII 13 followed by an ASCII 10 (Hex 0D/0A).

The two APIs are described in more detail below, followed by details for the possible requests and responses the make up the API.

TCP API

By default, Symmetry will listen for TCP communications on port 19850, but this can be changed in the Preferences. Symmetry can handle multiple connection at the same time, even from the same computer if necessary.

Generally there is no need to continually close and reopen a connection for each request. If you know that more requests will be sent in the next few moments, it is faster and more efficient to leave the connection open. There is, however, a time out period where Symmetry will close an open connection if no requests have been received during the time out period. By default, this is set to one hour, but it can be changed in the Preferences as well. Every time a request is received, the time out period starts over again for that connection. In either case, you should always close the connection once you know you are done with it to save resources.

Once you have established a connection with Symmetry, you send a properly formatted request followed by an end of line character (as explained above). The request will be processed and a result returned. More requests can be sent in the same fashion.

It is possible to send a subsequent request before receiving a result. All results returned will be in the same order that the requests were received.

An example communication flow would be like this:

- Establish a new TCP connection on the correct port.
- Send a request to Symmetry.
- Receive a result.
- Send another request.
- Receive a result.
- Close the connection.

Note that the proper holes must be opened in any fire walls that may exist between the client computer and the computer hosting Symmetry. You would normally want the computer hosting Symmetry to have a static IP address so you always know what address to connect on.

File API

When using this API, Symmetry will watch a specific folder (set in the Preferences) for new request files. Each time it scans the folder and finds new request files, it processes them by reading the request, creating a corresponding result file, and deleting the request file. The request file contents should only contain the request followed by the end of line delimiter (as explained earlier), exactly like the TCP API. The result file will be the same.

By default, Symmetry scans the folder for new requests every ten seconds. This can be changed in the Preferences. Once a scan cycle is complete, Symmetry will wait for this much time before starting a new scan cycle. This means that if a scan cycle takes two seconds, and the wait time is ten seconds, there is actually twelve seconds between scans.

Only one request is allowed per file. Any extraneous data after the first end of line delimiter will be ignored. If the request results in an error, the error will be returned in the result file in the same way as the error is returned by the TCP API.

Symmetry recognizes a file to be a request file by its extension which must be ".request". Similarly, your application can recognize result files by their extension which will be ".result". When multiple request files end up in the watch folder at the same time, it is important to know which result file corresponds to which request file. This is done with the filename itself. When Symmetry processes a request file, the result file it creates will have exactly the same name as the request file had, except that the extension will be changed.

Therefore, it is up to your application to ensure that request files always have unique names (because you don't know for sure when Symmetry will delete old request files) and that you keep track of the request filenames so that you can match up the results properly in your application. One suggestion would be to generate and use a UUID for each filename.

When Symmetry creates a result file, if a file already exists with the name it needs to use, Symmetry will attempt to overwrite the file. If it cannot, it will simply give up. However, if unique names are always used and the folder is kept clean, this will not happen.

Note that any files in the watch folder that don't have the ".request" extension will be ignored when Symmetry is processing the folder. However, it still takes CPU time for Symmetry to filter them out. Therefore, it is best to choose a watch folder that is free of other types of files.

As noted above, Symmetry will delete request files after they have been processed. It is your responsibility to delete the result files after you have received the result. This is important so that the watch folder stays clean and doesn't become to full (which can slow the system down).

It is possible for you to create the request file and only have the request partially written to the file when Symmetry first notices the file. Symmetry will not process the file until the end of line delimiter exists. You should apply the same logic when reading the result files.

An example communication flow would be like this:

- Create a file (or multiple files) with a request inside each file. Make sure they have the ".request" extension.
- Symmetry notices the files on the next cycle and processes them. The request files are deleted and result files with the same names are created.
- Your application watches for the result files and reads in the results. You match up the result to the request based on the filename without extension.
- The cycle continues.

Requests

This section details each possible request. Its function and parameters are explained, and then an example is given. In the examples, the request is given first in red and the result follows in blue. Note that "<CRLF>" is used to denote the proper end of line delimiter.

IsConnectionOK

This request can be used to test that an API is working. It works for either API.

This request has no parameters. You know there is a problem working with the API if an error result is returned or if no result at all is returned. If the connection is working, the result will be "OK".

```
IsConnectionOK()<CRLF>
OK<CRLF>
```

GetConvertedRFID

Symmetry does a variety of things to massage incoming RFID numbers into the 21 digit CCIA equivalent. This request allows you to test and/or use the converted numbers. It simply returns the massaged RFID number for the RFID number sent in the request. Just because it returns a number does not mean that the RFID number actually exists in the database.

```
GetConvertedRFID(LA CAN 000256233350)<CRLF>
000000000124256233350<CRLF>
```

DoesExist

This request takes one RFID and returns True if the RFID exists in Symmetry or False if it does not exist. Note that Symmetry only downloads RFID numbers from the CCIA if they have a birth date or a breed. Tags that are known to the CCIA, but do not have a birth date or breed will return False here because Symmetry will not know about them.

```
DoesExist(124000123456789)<CRLF>
True<CRLF>
```

IsAgeVerified

This request takes one RFID and returns True if the animal has a birth date. It returns False if it does not have a birth date or if it does not exist.

```
IsAgeVerified(124000123456789)<CRLF>
False<CRLF>
```

GetBirthDate

This request takes one RFID and returns the date of birth for the animal in YYYYMMDD format. If the animal does not exist or has no birth date, 00000000 will be returned.

```
GetBirthDate(124000123456789)<CRLF>
20070308<CRLF>
```

GetAge

This request takes one RFID and returns the age, in months, of the animal. If the animal does not exist or has no birth date, 0 will be returned.

```
GetAge(124000123456789)<CRLF>
27<CRLF>
```

GetAllAgeInfo

This request takes one RFID and returns the date of birth followed by the age followed by the date the previous information was registered with the CCIA database.

```
GetAllAgeInfo(124000123456789)<CRLF>
20070308,27,20070312<CRLF>
```

GetBreed

This request takes one RFID and returns the breed of the animal. If a breed was never attached to the animal or the animal does not exist the breed returned will be blank.

```
GetBreed(124000123456789)<CRLF>
AN<CRLF>
```

or

```
GetBreed(124000123456789)<CRLF>
<CRLF>
```

GetAllBreedInfo

This request takes one RFID and returns the breed of the animal followed by the date the breed information was registered with the CCIA database.

```
GetAllBreedInfo(124000123456789)<CRLF>
AN,20070312<CRLF>
```

IsExpressVerified

This request takes one RFID and returns True if the animal has been Express Verified. It returns False if not or if it does not exist.

```
IsExpressVerified(124000123456789)<CRLF>
True<CRLF>
```

GetAllExpressVerifiedInfo

This request takes one RFID and returns the Express Verification status of the animal followed by the date this information was registered with the CCIA database.

```
GetAllExpressVerifiedInfo(124000123456789)<CRLF>
True,20070312<CRLF>
```

Group_DoesExist

This request takes one or more RFIDs. For each RFID in the request, a True or False will be returned based on whether the animal exists.

```
Group_DoesExist(124000123456789,124000456756113)<CRLF>
True,False<CRLF>
```

Group_IsAgeVerfied

This request takes one or more RFIDs. For each RFID in the request, a True or False will be returned based on whether the animal has a birth date or not.

```
Group_IsAgeVerified(124000123456789,124000456756113)<CRLF>
True,False<CRLF>
```

Group_GetBirthDate

This request takes one or more RFIDs. For each RFID in the request, the birth date of the animal will be returned.

```
Group_GetBirthDate(124000123456789,124000456756113)<CRLF>
20070312,00000000<CRLF>
```

Group_GetAge

This request takes one or more RFIDs. For each RFID in the request, the age, in months, will be returned.

```
Group_GetAge(124000123456789,124000456756113)<CRLF>
27,0<CRLF>
```

Group_GetBreed

This request takes one or more RFIDs. For each RFID in the request, the breed will be returned.

```
Group_GetBreed(124000123456789,124000456756113)<CRLF>
An,<CRLF>
```

Group_IsExpressVerified

This request takes one or more RFIDs. For each RFID in the request, a True or False will be returned based on whether the animal is Express Verified or not.

```
Group_IsExpressVerified(124000123456789,124000456756113)<CRLF>
True,False<CRLF>
```

GetLastNightlyCheckStatus

Each night Symmetry downloads any new data from the CCIA database. This request can be issued remotely to see if the last download happened correctly or not. If Symmetry has not be able to download new data for multiple nights in a row, once the issue is fixed it will download all missing data from previous nights as well. So if this function returns "OK" then you know all the information is up to date. If an error is returned you know that something needs to be fixed.

```
GetLastNightlyCheckStatus()<CRLF>
OK<CRLF>
```

or

```
GetLastNightlyCheckStatus()<CRLF>
Error: some human readable error message<CRLF>
```

GetLicenseStatus

This request can be used to determine the license status of Symmetry. It will return OK unless Symmetry is not properly licensed.

```
GetLicenseStatus()<CRLF>
OK<CRLF>
```

or

```
GetLicenseStatus()<CRLF>
Error: some error message<CRLF>
```

Extra Information for the TCP API

The TCP API is capable of handling hundreds of requests per second on the same connection. However, in most cases, it will only be handling a request every few to several seconds.

There is logic built in to Symmetry so that whenever there are less that three to five requests per second, Symmetry will use very little CPU (usually less than one percent). One the request rate increases, Symmetry will begin using more CPU time very quickly to keep up. It has been tested at over 450 requests/second at about 60% CPU. Of course, this will vary depending on hardware and network.

More Information

If you need more help in figuring out how to communicate with Symmetry's APIs, please contact Cannon Smith via email (cannon@ssgfusion.com) or, if urgent, via phone (403-626-3236).