Using NexumAPI as a Service (NaaS) with a Validation PowerOn

October 2023

Libum

Overview

There are inherent limitations when developing within some PowerOn's. For example, a user is not able to leverage the FMPERFORM function while in Validation Specfiles. This document illustrates how a user could circumvent this limitation by executing a NexumAPI as a Service (NaaS) request to perform the file maintenance. For readers that don't already know, NexumAPI is a translation solution for SymXchange™ that provides the capability for developers to replace the native SOAP interface with a RESTful interface that is easier to use and manage. NaaS packages NexumAPI into a language agnostic microservice which also provides extended features.

A Simple Use Case

One use case is the simultaneous synchronization between fields on different records at the time of validation. Consider the scenario where a member comes into the credit union to update their email address, for some institutions this would need to be updated in both the Name Record as well as the Preference Record for Online Banking. If there's no set procedure in place for the credit union, it's likely that the representative will neglect to also update the email field in the Preference Record causing Online Banking to render outdated information.

What if we could synchronize the fields? Where no matter which field you update natively within Symitar, the other will reflect the same value instantaneously. We'll cover this example beginning in the next section.

Add NaaS Request to a Validation PowerOn

In this example, we are going to execute a NaaS request from a basic Validation PowerOn. Doing so will demonstrate how we can leverage the full suite of SymXchange™ operations to conduct any File Maintenance regardless of the PowerOn type.

Creating a Functional Validation PowerOn

First, let's create a boilerplate Validation PowerOn to be triggered on the Name Record using the following code block:

VALIDATE.NAME

```
validation
windows

target = name

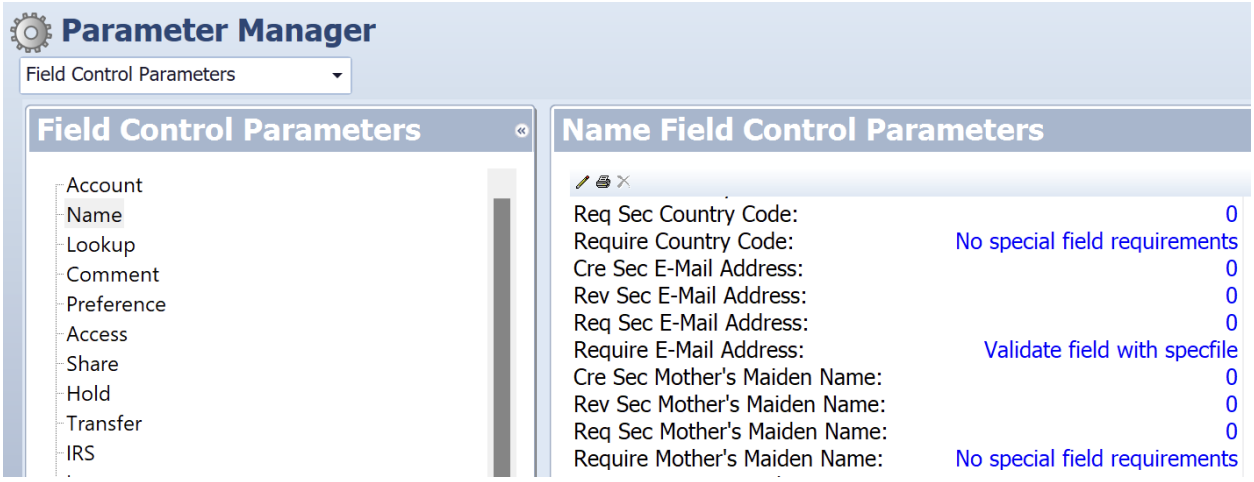
setup
  popupMessage(0, "Validating Name!")
end

print title=""
  suppressnewline
end

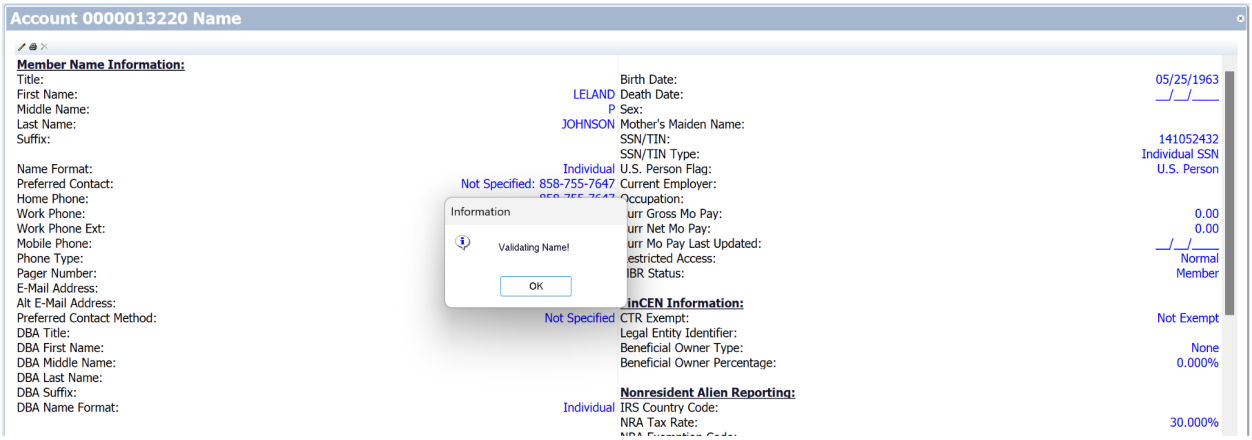
```

Save the file as **VALIDATE.NAME** and go ahead and install it for demand use. In order for this PowerOn to function within Symitar, we will also have to navigate to **Management > ParameterManager > Field Control Parameters** to enable particular fields for validation.

In this example, we will be monitoring the email address field for change, so find it within the Name record and set the *Require E-Mail Address* attribute to *Validate field with specfile* as shown below:



To test functionality, if you select an account in the Account Manager, navigate to a Name, and select the **E-Mail Address** field, you should see a popup message when (1) you first select the field for edit and (2) when you save it.



Adding the NaaS Request

Now that our Validation PowerOn is working as intended, we can update the code to include the NaaS request to synchronize the field with another. You'll notice we've added a few more definitions as well as encapsulated the functionality into a separate procedure:

```
6     define
7       true = 1
8       false = 0
9       i = number
10
11      nAPIfileName = character
12      nAPIfileNumber = number
13      nAPIfileError  = character
14      nAPIres = character
15      nAPIemail = character
16      nAPInameEmailField = number
17      nAPInameEmail = character
18      nAPIpreferenceLOC = number
19      nAPIpreferenceEmail = character
20      nAPIisAllFieldsUpdateSuccess = number
21      nAPIisEmailUpdated = number
22    end
23
24    setup
25      call synchronizeEmails
26    end
```

Now, within the **synchronizeEmails** procedure, we'll prepare a few variables and find the preference locator which will need to be updated:

```
[
/**
 * Synchronize Name Record updates to Email Address with Preference Record
 */
]
procedure synchronizeEmails
  if (account:number <> "") then
    do
      nAPIfileName = "novus.response." + format("9999", sysConsoleNum)
      nAPInameEmailField = 34
      nAPInameEmail = name:email

      [// Find preference locator to update]
      for each preference with (preference:type = 0)
        do
          nAPIpreferenceLOC = preference:locator
          nAPIpreferenceEmail = preference:emailAddress
        end
        until nAPIpreferenceLOC <> 0

```

If the preference locator and field maintenance type are correct, we'll do a few additional validations on the input given by the user (1) whether or not the email matches what currently exists in the preference record and (2) whether or not the input provided contains an '@'.

```
      [// Continue if preference locator was found & correct FM Type]
      if (@validateFMType <= 1 and nAPIpreferenceLOC > 0) then
        do
          [// Validate field and preference record]
          if ((@validateFieldNumber = nAPInameEmailField) and (@validateCharacterInput <> nAPIpreferenceEmail)) then
            do
              [// Validate and set the email input]
              if ((characterSearch(@validateCharacterInput, "@") > 0) or (@validateCharacterInput = "")) then
                do
                  nAPIisEmailUpdated = true
                  [// Handle new email value]
                  if (@validateCharacterInput = "") then
                    nAPIemail = " "
                  else
                    nAPIemail = @validateCharacterInput
                end
              else
                do
                  popupMessage(1, "An '@' is required for email addresses.")
                  validateFieldSet(name, email, nAPInameEmail)
                end
            end
        end

```

Finally, if all of the field checks have been met, we can leverage HTMLVIEWOPEN to make a silent Form URL Encoded API call to NaaS using the HTMLVIEWLINE directives. After the request, we'll have to read in the server response from a temporary Letter File that Symitar produces.

```
76          [// Send silent HTTP request to NaaS if email was updated]
77          if (@validateFieldNumber = 999 and nAPIisEmailUpdated = true) then
78            do
79              [// Construct silent HTTP request to NaaS instance]
80              htmlViewOpen(1)
81              htmlViewLine("URL=https://{{endpoint}}/nexum/{{sym}}/account/updatePreferenceByID")
82              htmlViewLine("VERB=POST")
83              htmlViewLine("HEADER=Content-Type: application/x-www-form-urlencoded")
84              htmlViewLine("HEADER=Authorization: Bearer {{token}}")
85              htmlViewLine("FORMFIELD=AccountNumber=" + account:number)
86              htmlViewLine("FORMFIELD=Credentials[AdministrativeCredentials][Password]={{password}}")
87              htmlViewLine("FORMFIELD=PreferenceLocator=" + format("##9", nAPIpreferenceLOC))
88              htmlViewLine("FORMFIELD=PreferenceUpdateableFields[EmailAddress]=" + nAPIemail)
89              htmlViewLine("WANTRESPONSEHEADERS=NO")
90              htmlViewLine("FILENAME=" + nAPIfileName)
91              htmlViewDisplay
92
93              [// Capture "sync variable", without, it will cause Symitar to crash]
94              suppress enterline(0)
95
96              [// Handle response]
97              fileOpen("LETTER", nAPIfileName, "READ", nAPIfileNumber, nAPIfileError)
98
99              if (nAPIfileError <> "") then
100               popupMessage(2, "Unable to read NaaS response: " + nAPIfileError)
101             else
102               do
103                 fileReadLine(nAPIfileNumber, nAPIres, nAPIfileError)
104                 if (characterSearch(nAPIres, "isAllFieldsUpdateSuccess") > 0) then
105                   nAPIisAllFieldsUpdateSuccess = true
106               end
107
108             fileDelete("LETTER", nAPIFileName, nAPIfileError)
109
110             [// Forward error response to Symitar UI]
111             if (nAPIisAllFieldsUpdateSuccess = false) then
112               popupMessage(2, "Missing 'isAllFieldsUpdateSuccess' in NaaS response.")
113           end
114       end
115   end
```

A Few Things to Remember

We wanted to include a few helpful remarks that might save you some time or answer questions you might be having.

1. **OnDemand PowerOn** - remember, every change you make to an OnDemand PowerOn must be installed and the account refreshed before you'll see those changes reflected.

2. **Abstraction** - if you plan to have several procedures which perform field checks and make requests to NaaS, a good design choice is to abstract out the HTTP call into a separate parameterized procedure.

3. **Inline Credentials** - this example is just a proof of concept (plain text Admin Credentials is less than ideal), we recommend exploring more secure ways to authenticate the request such as tokenization or simple obfuscation through encoding a DATAFILE reference.

Resources

For more information, please contact us at development@libum.io or ask whoever gave you this paper about how to join our community Discord server.