NICOF

Non-***I***nvasive ***CO***mmunication ***F***acility

Version 0.5.0 for VM/370 R6 SixPack 1.2

Dr. Hans-Walter Latz, Berlin, Germany, 09.12.2012

A (very) informal guide to NICOF usage

WARNINGS:

This software is delivered as-is with no promise or commitment to be usable for any particular purpose.
Use it at your own risks!
The NICOF software and documentation has been written by a hobbyist for hobbyists and should not be used for any important or even critical tasks.

NICOF is work in progress and its current implementation may differ from this document.

License:

Contents

1 Introduction

1.1 Overview

NICOF (***N***on-***I***nvasive ***CO***mmunication ***F***acility) gives VM/370 running on Hercules the ability to actively communicate with the outside world of the simulated mainframe.

More precisely, it allows *inside* processes (programs running in VMs under VM/370) to exchange data packets with *outside* services implemented in a Java process. The terms *inside* and *outside* will (attempted to) be used consistently to identify the programs/processes *inside* VM/370 resp. the processes *outside* Hercules implementing the functionality communicated with.

The active part of the communication is in all cases the *inside* process (a client program under CMS), which initiates each data transmission by sending a request (data packet with metadata) to the *outside* service implementation, which processes the request and sends back a response (data packet with metadata) to the client.

As the term "non-invasive" indicates, NICOF does not need modifications or extensions to VM/370 R6 or to Hercules. It works with the on-board internal and external communication means which are part of VM/370 or Hercules. However, a Java Runtime Environment is needed to provide the outside infrastructure and services.

1.2 How does it work?

In an end-to-end view of a single NICOF communication…

- The communication partners are the client CMS program on the inside in a client-VM and a service instance on the outside in the Java-process.
- In the middle between these 2 communication partners is a central proxy-VM where the CMS NICOF proxy program runs. This proxy program is called the *inside proxy*.
- The client-VM communicates with the proxy-VM through VMCF.
- The outside Java program connects to the proxy-VM via a DIAL-ed 3270 terminal connection, the data is exchanged between them through screen writes (inside → outside) and simulated user inputs (outside → inside) initiated by the NICOF proxy program through SIO operations to the attached (pseudo-) 3270 terminal.
  This Java proxy program is called the *outside proxy*.
  This Java program can run "anywhere", i.e. on any computer from where it can connect to the Hercules emulator. The outside program can use the local resources of this computer when providing services to the inside, for example the file system, TCP/IP sockets or databases.
- The inside and outside proxies use a more or less elaborate handshaking to synchronize and confirm the bidirectional data transfer over the simulated 3270 terminal connection.

The inside proxy (the central proxy-VM) can be accessed at the same time by several clients-VMs (CMS users) and relays the requests to the outside proxy along with metadata which will allow to transmit the responses from the outside process back to the correct client-VM and to relate them there to the correct requests.

In an API oriented view, a request is on the lowest level a data packet with 0 to 2048 bytes and additionally 2 fullwords as additional metadata sent along with the data packet (so called userwords

at the discretion of the client-service protocol for sequencing requests, selecting subservices or functions etc.). At this lowest level, NICOF is protocol and service agnostic, i.e. NICOF works exclusively as data transporter. This lowest level is called ***Level-0***. In the configuration of the outside Java process, the Java class is specified which will receive and process the Level-0 requests. The implementation of the class must of course match the expectations of the client programs sending the requests regarding the meaning of the packets and userwords sent forth and back as well as the functionality requested.
A dedicated instance of this class is instantiated for each client-VM (VM/370 user) at the first request; this instance exclusively handles all requests from this VM.
At the Level-0, NICOF supports using different proxy-VMs, each connected to a dedicated outside Java process, each providing a separate set of Level-0 services. The Level-0 API routines accept the name of the target proxy-VM. If none is provided, the default proxy-VM NICOFPXY is addressed.

As part of NICOF, a default Level-0 service is delivered which allows to provide several services in parallel, using a unified programming model both on the inside and the outside. This ***Level-1*** service infrastructure uses one of the userwords to transport the information about the service to be invoked (service-id) and the function to be called on that service for the request (command) and the return code for the response. In the configuration of the outside Java process, a list of Level-1 service classes with their associated service name is specified. For each client-VM issuing requests to the outside, a separate set of instances of these service implementations is instantiated on the first request.
On the inside, the Level-1 infrastructure allows to resolve a service name to a service-id and to invoke a command on a service, passing a data packet and a control word (the last remaining userword) and receiving a data packet along with a returncode and a controlword. Furthermore, the Level-1 infrastructure supports an STDIO-like stream operation like paradigm to the data exchange between the inside and outside.
At Level-1, only the default proxy-VM is currently supported. This is sufficient as multiple services can be provided at Level-1.

The NICOF proxy program and the APIs are implemented mainly in C, supported by an assembler module allowing to process interrupts and issue SIO operations from a C program.

1.3 What is available?

The communication infrastructure core of NICOF is now believed to work without loss of packets or spontaneously stopping to work (with the emphasis on “is believed”).

Besides the NICOF communication core with the inside and outside proxy implementations, NICOF comes with the following services:

- NHFS – ***N***icof ***H***ost ***F***ile ***S***ystem
  This Level-1 outside service allows to access files and directories on the outside host (OS where the outside proxy runs) from CMS with the NHFS command. Accessing means:
  - list and create directories on the outside host
  - transfer files between the outside host and the CMS A-disk.
  Accessing files is intentionally restricted to files and directories following naming rules similar to the rules CMS files on a minidisk. Furthermore, only files and directories in the configured exchange area (base directory) can be accessed with NHFS.

- Load and mass test
  This Level-0 testbed was used to develop and verify the communication mechanisms of NICOF. The outside service simply echoes the request sent by the CMS client with a few changes in the data packet.

Inside proxy can be run interactively or in background through AUTOLOG. The outside proxy is currently started manually with a script (but could be made part of the VM/370 start script).

NICOF was developed and tested on MS Windows Vista 32 bit and also tested on Windows7 64 bit with Hercules 3.07 and using a Java 1.6 SDK. It is known to somehow work under Ubuntu Linux 9.10 with the delivered Unix files (with “somehow” meaning: does not crash on the first usage).

1.4 Performance

The load and mass test EXEC (see 4) also allows to make simple throughput performance measurements. It allows to have a defined number of 2048 byte packets sent to the outside and returned from there. The time measurement is done at the EXEC level with simple QUERY TIME calls allowing to compute the time elapsed for specified number of packet roundtrips.

The measurements on 2 Windows systems gave the following results (based on the CPU of the machines) for the packet roundtrips (i.e. KByte/s means Kbyte-*roundtrips* per second, SIO/s and MIPS as indicated by the S/370 screen of the Hercules console):

➔ Core2 Duo @ 2.4 GHz
  ~ 138 KByte/s, ~ 450..540 SIO/s average at ~ 16.5..17.5 MIPS
➔ Core i7-3770 @ 3.40 GHz
  ~ 430 Kbyte/s, ~1600..1700 SIO/s at 42..45 MIPS

1.5 Which other services could be provided?

These are just ideas, limited only by fantasy and available time to implement:

- Access to databases outside VM/370: the Level-1 outside service would use the standard JDBC API to access arbitrary relational (or other types of) database, forwarding the SQL statements coming from the inside to the database and transmitting the result (result sets or the like) to the inside.
- Client access to internet services, for example FTP-client: the Level-1 outside service would use Java client libraries for internet services to provide a high-level access to the internet services.

1.6 What are the current limitations?

The data exchange between the inside and outside through the 3270 terminal to which the outside proxy is dialed occurred originally in binary mode in both directions. This is theoretically not a problem, as the binary telnet protocol used in 3270 communication allows to escape the 0xFF special character used to initiate the telnet handshaking.

During mass tests, there was the problem that sometimes (very sporadically) in the data transfer outside to inside (i.e. sending a “screen” content), the escape 0xFF (sent as 0xFF 0xFF) seems to be interpreted as “record end” 3270 telnet sequence (0xFF 0xEF), which prematurely terminates a response (bad: data loss) and starts a new transmission , doing some internal handshake with the next byte as handshake token (worse: communication typically locks up). It could not be determined if the related bug was in the internal implementation of the 3270 binary encoding (seemed to be OK), if it is in the java socket implementation (improbable as heavily used in productive systems) or if it is in the Hercules 3270 terminal part (improbable as heavily used).

For this reason, the data transmission outside to inside was shifted to an encoding scheme avoiding 0xFF characters: 7 bytes are sent with the high-bit reset, followed by an 8-th byte having the high-bits of the previous 7 data bytes. Although the data transfer works well since this encoding is used, this slowed down the throughput by at least 1/3 or maybe a 1/2 (increasing CPU usage and reducing the SIO/s).

Other problems may arise from the fact that not all functionalities of the NICOF APIs have been tested so far. For example: the API allows to have several requests working in parallel, sending several requests and waiting for the first (the next) response to come back. This may work, but is not tested…

And while an API chapter is missing in this document, the only API documentation is the source code (having some comments for the procedure and method headers). Sorry about that…

2 Installing a test environment

The easiest way to explore NICOF is to start with a fresh VM/370 Sixpack 1.2 installation (adding MECAFF is not required, although may be helpful).

If an existing VM/370 system is used, a backup of the relevant files is surely a good idea before experimenting with NICOF (at least the shadow files of the VM/370 disks and the Hercules configuration file for the system).

The following explanations are for a MS Windows OS hosting the Hercules & VM/370 system. Using NICOF on Linux/Unix systems should be similar when using the provided files (shell scripts) equivalent to the Windows files.

2.1 Prerequisites

To use NICOF, a Java runtime version 1.6.0 (Java 6) is required; a newer runtime 1.7 (Java 7) may work, but this has not been tested.

The inside NICOF tools, specifically the CMS NHFS program, and the current build procedure rely on the GCC native CMS runtime available in VM/370 R6 SixPack 1.2. It may be possible to use recompile the NICOF programs under SixPack 1.1 as the GCC native CMS runtime is also available there (untested).

2.2 Content of the delivery package

The file `nicof-0.5.0.zip` accompanying the documentation contains the following directories and files:

disks
Extension of the disks-directory of a standard VM/370R6 Sixpack 1.2 installation.
disks/vm3350-z.ncfdsk.7f0.cckd
A 3350 disk pack preloaded with the NICOF users, sources and binaries. As this data does not fill up a 3350 disk, there is enough spare space for new users of the VM/370 system.

nicof
The directory with all runtime items needed to use NICOF.
nicof/nicof-maint-ncfdsk_direct.aws
AWS-tape with a DIRECT-file extract defining the VM-users in the NICOF disk pack. These are: • NICOFSRC a pseudo-user providing the minidisk with the NICOF sources and binaries • NICOFPXY the user for the inside proxy VM • CLIENT01, CLIENT02, CLIENT03 CMS users for NICOF All these users have their username as password.
nicof/nicof.jar
The Java-jar (binary) of the NICOF outside proxy including the default services delivered.
nicof/run-nicof.cmd nicof/run-nicof.sh nicof/default.properties
The scripts and configuration file for running the outside proxy with Level-1 services (currrently NHFS and some test services used for development)
nicof/run-nicof-level0echo.cmd nicof/run-nicof-level0echo.sh nicof/level0_echo.properties
The scripts and configuration file for running the outside proxy with the Level-0 “Echo” service used for the mass and load tests (and for development).
nicof/nicof_logging.properties
The configuration file controlling the logging of the outside proxy to the console.
nicof-src
Directory containing all sources of NICOF.
nicof-src/nicof-nicofsrc_192-files.aws
AWS-tape with the sources and precompiled CMS MODULEs for the inside part of NICOF.
nicof-src/nicof-java-src-0.5.0.zip
The Java sources of the outside proxy and the available services on the outside.
nicof-src/nicof-nicofpxy.aws nicof-src/nicof-client01.aws nicof-src/nicof-client02.aws nicof-src/nicof-client03.aws
AWS-tapes with the local CMS files (PROFILE and other EXECs) for the CMS users NICOFPXY, CLIENT01, CLIENT02, CLIENT03.
HerculesConfiguration_NICOF-pack-win.txt HerculesConfiguration_NICOF-pack-unix.txt
Text files with the configuration addition to add the NICOF-3350 disk pack to the Hercules configuration file for VM/370. The disk pack is added as (real) device 7F0 to the Hercules hardware.

2.3 Unpacking the files

The archive `nicof-0.5.0.zip` should be unpacked in the base directory of the VM/370 SixPack installation (where the directories disks, io, batch can already be found).

This will add the new directories `nicof` and `nicof-src` to the VM/370 installation and the NICOF disk pack to the `disks` directory.

2.4 Extending the Hercules configuration for VM/370

Add the content of the file `HerculesConfiguration_NICOF-pack-win.txt` to the Hercules configuration file for VM/370, typically the file `sixpack.conf` (for a plain Sixpack 1.2 installation) or `sixpack+mecaff.conf` (if MECAFF is also installed). Appending the new configuration lines at the end of the .conf-file is sufficient, it is not necessary to place the lines at a specific place in the file.

This works if the real device 7F0 is not already in use. If this is the case, any other device defined as 3350 in DMKRIO can be used instead of 07F0 in the new configuration file (the .cckd file needs not to be renamed, although it has the device address in its name).

2.5 Setup in VM/370

When Hercules is started with the extended VM/370 configuration file (see above), the new disk pack will already be available in the system. However, the structure of this disk pack (which CMS minidisks are in which cylinders) must be made known to the system.

For this, the tape file `nicof/nicof-maint-ncfdsk_direct.aws` has a DIRECT file fragment defining new users and their minidisks on the NICOF disk pack. To setup NICOF, use the following steps:

- Load the tape with the following command in the Hercules console:
  ```
  devinit 480 nicof/nicof-maint-ncfdsk_direct.aws
  ```
- Logon as user MAINT
- Attach and load the tape:
  ```
  ATTACH 480 TO * 181
  TAPE LOAD
  ```
  (this will create the file `NICOF DIRECT` on the disk)
- Append the content of the file NICOF DIRECT to the directory file defining the current users on the system. This can be done with the preferred editor or for example with the following commands (with *yours* being SIXPACK on a fresh VM/370 SixPack 1.2 system):
  ```
  RENAME yours DIRECT A PRE_NCF DIRECT A
  COPY PRE_NCF DIRECT A NICOF DIRECT A yours DIRECT A
  ```
- Activate the new directory:
  ```
  DIRECT yours DIRECT
  ```
  (this will add the users NICOFSRC, NICOFPXY, CLIENT01, CLIENT02 and CLIENT03 to the CP directory)

2.6 Starting the NICOF subsystem

The inside proxy can be started

- either interactively by logging on as NICOFPXY and running the exec `RUN-PXY`
- or in the background by doing an AUTOLOG for the user NICOFPXY; this can be done in the Hercules console by issuing
  ```
  /autolog nicofpxy nicofpxy
  ```
  or by logging in as user MAINT and issuing the same command from this user:
  ```
  AUTOLOG NICOFPXY NOCOFPXY
  ```

The outside proxy is started by

- changing to the `nicof` directory (see above)

- and invoking the cmd file `run-nicof.cmd`
  (or alternatively `run-nicof-level0echo.cmd` if the load tests are to be run, see 4)

There is no specific startup sequence (which of the proxies to start first), as the outside proxy will try to connect to the inside proxy all 3 seconds until it can successfully DIAL to the NICOFPXY user and start up the communication. The outside proxy will also try to reconnect if the connection is lost (the inside proxy crashes or disappears in some other way)

When starting the outside proxy before the VM/370 is started, the following message sequence can appear on the NICOF console window:

```
main INF: CommProxy: Connecting...
main INF: CommProxy: Unable to setup Proxy connection to VM/370-Host 'localhost', Port 3270
main INF: CommProxy: Proxy-VM 'NICOFPXY' not present or not accepting DIALs
main INF: CommProxy: Connected to Proxy-VM
```

The second line comes if Hercules is not started, the third comes when the proxy-VM is not running or the proxy program is not running in this VM.

Of course, some of the messages will not appear if the outside proxy is started at a later point of the VM/370 startup procedure, so possibly only the message “`Connected to Proxy-VM`” may appear.

2.7 Accessing the NICOF tools disk

The NICOFF tools (the MODULEs for the inside proxy program and the NHFS command) as well as the CMS sources for NICOF (including a script to build the NICOF programs) are located on the minidisk 192 of the user NICOFSRC.

This minidisk must be linked and accessed to use the NICOF tools. The users on the NICOF disk (NICOFPXY and CLIENT*) already do this in their PROFILE EXEC.

2.8 Configuration of the outside proxy

The connection parameters and service configuration are defined in the properties file passed to the outside process. If no properties file is given, the file `default.properties` is used.

(see the comments of the file `default.properties` for available configuration options)

2.9 Installation variants

Running the inside proxy can be initiated at system startup time by extending the configuration of the AUTOLOG1 user to also AUTOLOG the NICOFPXY user.

Similarly the startup script for the complete VM/370 system can be extended to also start the NICOF outside proxy by calling the `run-nicof.cmd`-script in the `nicof` directory. However, this script must be called in the `nicof` directory (to be able to find the own configuration files) and must be started in the background.

If all users on VM/370 will use the NICOF tools (currently NHFS), these tools can be copied to the disk 19E which is normally accessed with mode Y/S. Care should be taken to copy to MODULEs to this disk with the file mode Y2 to be usable, as the minidisk is normally accessed in Read/Only mode.

3 Using NHFS

NHFS allows to access a subset of the files on the computer running the outside proxy. This service is available when the outside proxy is started with the standard Level-1 configuration (using the `default.properties` configuration).

3.1 Naming conventions and restrictions for outside files

The NHFS service intentionally imposes some restrictions to the outside files (on the computer where the outside proxy runs) to be visible to the inside.

First: only files under a user-specific base-directory can be accessed. This user-directories are located in the base-directory of the NHFS service, which is configured for the service, and are created when a inside VM accesses the NHFS service for the first time.

Second: the name components of the files and directories below a user's base-directory must comply with the CMS naming conventions for files. The name parts may be max. 8 characters long with the same allowed characters as CMS filenames and must be named like *`filename.filetype`*. This allows to transfer the files between the in- and outside without having to specify 2 file-ids, as the same file-id is used to identify both the inside and the outside files. The name of directories must also be max. 8 characters long and with the same allowed characters as CMS filenames. Files and directories not complying with these rules are simply ignored by the NHFS service and are simply invisible to the inside.

3.2 Configuring the outside service

The base directory for the NHFS service is defined in the outside proxy configuration and must be a directory accessible and writable by the java outside proxy process. See the file `default.properties` for the NHFS configuration parameter.

3.3 NHFS subcommands

The CMS NHFS command provided with NICOF allows to access files and directories on the outside in the invoking user's base directory.

The format of the NHFS invocation is:

```
NHFS <subcommand> [ id-tokens ] [ ( options ]
```

The *id-tokens* parameter is a sequence of name tokens (max. 8 characters in the CMS file naming convention). Depending on the subcommand, the *id-tokens* specify either a file (*`fn ft`*) or a directory path from the user's base-directory or both (in which case the file-id comes first).

NHFS has the following subcommands:

- `list [ dir1 [ dir2 [ …] ] ]`
  → list the files and directories in the specified directory *`dir1`*/*`dir2`*/…
  → if invoked without directory names, the content of the user's base-directory.
- `mkdir dirname [ dir1 [ dir2 ... ] ]`
  → create the new subdirectory `dirname` in in *`dir1`*/*`dir2`*/… in the user's base-directory
- `type fn ft [ dir1 [ dir2 ... ] ]`
  → type the content of the outside file `fn.ft` on the console;
  → the file is located in *`dir1`*/*`dir2`*/… in the user's base-directory

- `put fn ft [ `*`dir1`*` [ `*`dir2`*` ... ] ] [ ( REPLACE ]`
  `putbin fn ft [ `*`dir1`*` [ `*`dir2`*` ... ] ] [ ( REPLACE ]`
  → copy the inside (CMS) file `fn ft` to the outside file `fn.ft`
  → the file will be written in *`dir1`*/*`dir2`*/… in the user's base-directory
  → the subcommand `put` transfers the file in text mode, `putbin` in binary mode
  → in text mode, the EBCDIC to ASCII translation is done for the bracket EBCDIC charset
  → if the target file already exists, the REPLACE option must be given to overwrite it
- `get fn ft [ `*`dir1`*` [ `*`dir2`*` ... ] ] [ ( [REPLACE] [LRECL `*`len`*`] [RECFM `*`x`*`] ]`
  `getbin fn ft [ `*`dir1`*` [ `*`dir2`*` ... ] ] [ ([REPLACE] [LRECL `*`len`*`] [RECFM `*`x`*`]]`
  → copy the outside file `fn.ft` to the inside (CMS) file `fn ft`
  → the file will be looked for in *`dir1`*/*`dir2`*/… in the user's base-directory
  → the subcommand `get` transfers the file in text mode, `getbin` in binary mode
  → in text mode, the ASCII to EBCDIC translation is done for the bracket EBCDIC charset
  → if the target file already exists, the REPLACE option must be given to overwrite it
  → the options `LRECL` and `RECFM` specify the format of the CMS file created
  → *`len`* must be 1..255, *x* can be V or F; the defaults are LRECL 80 and RECFM V

4 Doing the load tests

To run the load tests, the NICOF sources on the CMS side must first be completed, as the necessary TEXT files are not part of the delivery.

4.1 Compiling the CMS side sources

To recompile the NICOF CMS sources:

- Log on as user CMSUSER (for example)
- Link the NICOF source minidisk and replace disk A with this minidisk:
  ```
  LINK NICOFSRC 192 299 MW
  RELEASE A
  ACCESS 299 A
  ```
  (MW-password: MULT)
- Build NICOF :
  ```
  NCOMP [ ALL ]
  ```
  (specifying ALL will first delete all TEXT files for force a rebuild of all modules, as NCOMP normally only compiles modified files)

4.2 Running the tests

First it must be ensured that the outside proxy runs the Echo service.

If outside proxy is currently running with the default configuration (with the Level-1 servies, mainly NHFS), shutdown VM/370 and stop the external proxy. Then restart VM/370, autolog the NICOFPXY user and start the outside proxy with the Level-0 echo service with the script
```
run-nicof-level0echo.cmd
```
in the `nicof` directory (see above).

To start the load test, log on as one of the users CLIENT01, CLIENT02 or CLIENT03 and enter

`RUN-CLNT `*`count`*

with `count` being the number of 2048 byte packets to send to the outside proxy and to receive back.

The load test can be run simultaneously from all 3 CLIENT-users. The `RUN-CLNT EXEC`s of the 3 users wait a different time amount until the test really starts (CLIENT01: 3 seconds, CLIENT02: 2s and CLIENT03: 1s), so it is easy to make the test start simultaneously in 3 terminals.