

Long Island System Users Group



Lab Exercise: **Exploring Open Source Solutions on IBM i**

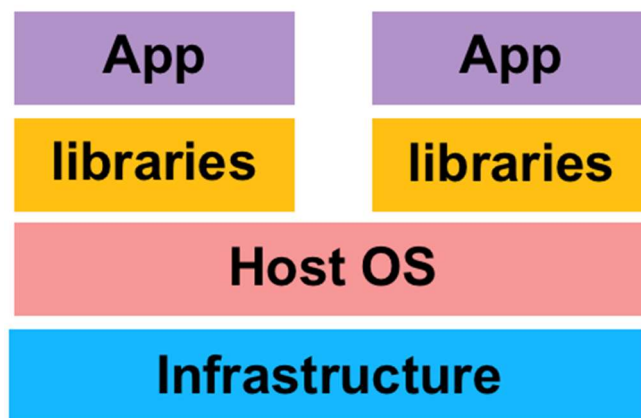
Speaker Name: **Erwin Earley**
(erwin.earley@roguewave.com)

Introduction:

Open Source Solutions (OSS) are in vogue for businesses large and small. You might be surprised at the wealth of Open Source Solutions that are available to leverage directly on IBM i. This workshop provides the opportunity to get your feet wet with OSS on IBM i. The self-paced exercises allow you to explore open source programming languages like python, php, and node.js as well as management of open source packages.

1. Establishing a container

Containers allow for the isolation of directories such that different users can have different environments that are dedicated to their usage. IBM i has the '`chroot`' capability available to enable establishing of what are often referred to as a 'chroot jail' – an isolation of directories and files. The following diagram provides a high-level view of chroot jail:



In the above example there are two 'chroot jails' that sit on top of the host operating system and the system infrastructure. This first lab will walk through the creation of chroot jail.

NOTE: Support for Open Source Solutions (OSS) / packages is provided through the OSS bootstrap which installs a number of utilities and packages and also establishes a package repository which is a definition for a location of package files. The boot strap has already been installed on the system being used for the lab. Contact the instructor if you would like a lab that covers the steps for setting up the OSS boot strap that you could use on your system to continue your exploration of open source on IBM i.

Section 1. Establishing an SSH connection to the lab system

The first step is to establish an SSH connection to the lab system.

1. Use the credential and system information in the lab token to establish the ssh connection using an SSH client (typically PuTTY on Windows, ssh command on MacOS)

NOTE: This connection will be used throughout the rest of this lab

Section 2. Install and setup `chroot`

The `chroot` capability on IBM i is provided through the `ibmichroot` package. This package has already been installed on the lab system.

Once `chroot` has been installed, chroot jails can be configured for individual users.

NOTE: A self-guided lab for installing the `ibmichroot` package on your system is available. Ask the instructor if you are interested in a copy of this lab.

- __ 2. The '`chroot_setup`' command will be used to configure the chroot jail. The command takes as arguments the directory that will be used as the chroot jail as well as a type indicator and an indication for language support. For the lab enter the following command:

```
chroot_setup /QOpenSys/<team> minimal nls
```

- Where `<team>` is replaced with the team name from your token

Observe that a lengthy list of diagnostic messages is output. This process will take several minutes.

Software packages can be installed in the chroot-jail. This allows different users to have different sets of applications (and configurations) and for those to be distinct from each other. Several packages will be installed in the chroot-jail through the remainder of this workshop.

- __ 3. Prior to entering the chroot jail let's install the '`bash`' package into the container. `bash` is a popular shell (command interpreter) in the Linux/Open-Source space. The `yum` command will be used to install the package (ACS can also be used to install Open Source Packages in a chroot container; however, ACS is not available on the workstation laptops). Enter the following command:

```
yum --installroot=/QOpenSys/<team> install bash
```

- o Where `<team>` is replaced with the team name from your token

NOTES:

- The '`installroot`' option instructs `yum` to use the specified directory as the root directory rather than the default root directory
- The '`install`' keyword instructs `yum` to perform an installation
- '`bash`' indicates the name of the package to install

One final item before entering the jail. The `chroot_setup` command doesn't create a home directory in the chroot jail. Prior to our entering the jail, let's create a home directory

- ___ 4. The next step is put ourselves in the chroot jail. This is done with the 'chroot' command. Enter the following command:

```
chroot /QOpenSys/<team> /QOpenSys/usr/bin/bash
```

- o Where `<team>` is replaced with the team name from the token

NOTES:

- The first argument is the directory that has been configured as the 'chroot jail'. This directory will become the root (/) directory for the user as a result of running the `chroot` command
- The second argument represents the program to execute. Typically, it is a shell so that you can traverse and work within the chroot jail. The path represents the path within the chroot jail

You may have noticed when entering the container an error was displayed indicting that the home directory was not found. This is because the home directory is not created as part of the `chroot_setup` process. The following two steps take advantage of the `$HOME` environment variable to create the home directory and to ensure that the ownership of the directory is properly set.

- ___ 5. Enter the following command to create the home directory.

```
mkdir $HOME
```

- ___ 6. Enter the following command to ensure the correct ownership of the directory created in the previous step:

```
chown <team> $HOME
```

- ___ 7. When the command is complete take a look at the current working directory with the following command:

```
pwd
```

Notice that the current working directory is root (/) – this is actually the /QOpenSys/<team> directory within the overall file system but for the scope of the running job (chroot) the user is limited (jailed) to the indicated directory since it's not possible to traverse above the root (/) directory.

- __ 8. Take a look at the directory listing with the following command:

```
ls -l
```

The output will resemble the following:

```
total 208
drwxr-sr-x    6 student0 0      8192 Apr 30 09:10 QOpenSys
lrwxrwxrwx    1 student0 0        34 Apr 30 09:04 bin -> /QOpenSys/usr/bin
drwxr-sr-x    4 student0 0      8192 Apr 30 09:04 dev
drwxr-sr-x    3 student0 0      8192 May  1 19:35 home
lrwxrwxrwx    1 student0 0        34 Apr 30 09:04 lib -> /QOpenSys/usr/lib
drwxr-sr-x    5 student0 0      8192 May  1 20:04 node_modules
-rw-r--r--    1 student0 0     10803 May  1 20:04 package-lock.json
lrwxrwxrwx    1 student0 0        36 Apr 30 09:04 sbin -> /QOpenSys/usr/sbin
drwxr-sr-x    5 student0 0      8192 May  1 20:02 tmp
drwxr-sr-x    2 student0 0      8192 Apr 30 09:04 usr
lrwxrwxrwx    1 student0 0        26 Apr 30 09:04 var -> /QOpenSys/var
```

A few things to make note of:

- The QOpenSys directory here is not the /QOpenSys directory in the IFS – rather this is the QOpenSys directory that was created in the chroot container when the chroot-setup command was executed (the reality is that this directory is the /QOpenSys/<team>QOpenSys directory – boggles the mind until you get used to it. ;-)
- Likewise, the home directory here is not the user's home directory outside of the chroot jail which is typically located in /home/<team>. When a 'cd' is performed in the chroot container the user will be placed in the home directory in the container – it needs to be understood that the files in this home are not the same as the files located in the users non-chroot home directory.

Let's take a quick exploration of some of the features of the 'bash' shell:

- ___ 9. The shell maintains a history of previous commands that have been executed. The 'history' command can be used to display a list of those commands:

```
history
```

Note that the commands are shown preceded with a number. Any command in the history list can be re-executed simply by entering ! followed by the number from the command history list.

Additionally, previously executed commands can be recalled through use of the up-arrow key. This will allow you to scroll through the list of previously executed commands. A command can be executed simply by pressing <ENTER> after the command has been recalled.

- ___ 10. Another powerful feature of the bash shell is file-name completion. File-name completion is accomplished by pressing the <TAB> key after entering a portion of a file-name, at this point the shell will complete as much of the name as possible while remaining unique. If there are multiple names that match what has been entered, then pressing the <TAB> key twice will show those matches. To see this in action try the following:

```
type ls /QO
```

```
Press <TAB>
```

Notice that the shell completes /QOpenSys/

```
Now press <TAB> twice
```

Notice that the shell provides a list of the items under /QOpenSys/

```
At this point simply press <ENTER> to execute the ls command.
```

Section 3. Package Management

Package management can be accomplished via command-line (as shown with the yum command in a previous step) as well as with Access Client Solutions. Since ACS is not installed on the workshop laptops we will use CLI commands in this section to prepare the chroot container for the remainder of the lab.

- __ 11. Exit the chroot jail:

```
exit
```

- __ 12. The 'provides' option to the yum command can be used to determine which package provides a particular command. One of the commands that needs to be installed in the chroot container is the `nano` (editor) command. Issue the following command to determine which package needs to be installed:

```
yum provides nano
```

The system should respond with something similar to the following:

```
ibm | 2.2 kB 00:00
nano-2.9.0-0.ppc64 : Small and friendly text editor
Repo : ibm
```

The output indicates that the command 'nano' is provided by the package 'nano-2.9.0.0.ppc64' from the ibm repository. The format of the package name is <package name>-<version>.architecture. For installation purposes only the package-name is required.

- __ 13. Install 'nano' in the chroot container with the following command:

```
yum --installroot=/QOpenSys/<team> install nano
```

- __ 14. A number of additional packages will be needed for the remainder of the lab. Run each of the following install commands to install the packages in your cohort container:

```
yum --installroot=/QOpenSys/<team> install python3
yum --installroot=/QOpenSys/<team> install nodejs
yum --installroot=/QOpenSys/<team> install git
yum --installroot=/QOpenSys/<team> install curl
yum --installroot=/QOpenSys/<team> install python3-pip
yum --installroot=/QOpenSys/<team> install python3-ibm_db
```

NOTE: One could make use of the command-recall (up-arrow) provided by the bash shell to simply recall the command and change the package name.

NOTE: Another approach to installing the above packages into the chroot container would have been to install rpm and yum into the container (`yum -installroot=<container> install rpm yum`) and then performed the installation inside of the container. Either method will provide the same result.

Before leaving this portion of the exercise let's take a look at a few more package management commands.

- ___ 15. One of the features/components of a package is package information including a description of the package, the repository the package resides in, the architecture, and so forth. Information on a package can be displayed with the 'info' option to the yum command. Execute the following command:

```
yum info nodejs
```

The following should be output:

```
Available Packages
Name           : nodejs
Arch           : ppc64
Version        : 8.9.3
Release        : 0
Size           : 22 M
Repo           : ibm
Summary        : Node.js JavaScript Programming Language
URL            : https://www.nodejs.org
License        : MIT
Description    : Node.js? is a JavaScript runtime built on
                  Chrome V8 JavaScript engine. Node.js uses an event-
                  driven, non-blocking I/O model that
                  : makes it lightweight and efficient.
```

- ___ 16. A list of packages available for installation can be obtained with the following command:

```
yum list available
```

A list of packages that are available (and not currently installed) will be displayed:

Available Packages		
activemq.noarch	5.11.1-1	ibm
autoconf.noarch	2.69-1	ibm
automake.noarch	1.15-1	ibm
bash.ppc64	4.4-1	ibm

Other options to the 'yum list' command include:

- `installed` → provide a list of the packages installed on the system (or container)
- `all` → show all packages, both installed and available
- `kernel` → show the list of packages related to the kernel

__ 17. The 'check-update' function of yum can be used to display a list of installed packages that have updates available. Enter the following command:

```
yum check-update
```

Output similar to the following should be output:

bash.ppc64	4.4-1	ibm
libgcc_s1.ppc64	6.3.0-24	ibm
liblzma5.ppc64	5.2.3-3	ibm
libopenssl1_0_0.ppc64	1.0.2q-0	ibm
libutil1.ppc64	0.3-99	ibm
python2.ppc64	2.7.16-1	ibm
python2-rpm.ppc64	4.13.0.1-17	ibm
rpm.ppc64	4.13.0.1-17	ibm
yum.noarch	3.4.3-17	ibm

To install updates the 'yum upgrade <package>' command would be used. Please don't do this in the lab environment

__ 18. A history of actions taken by yum can be displayed with the following command:

```
yum history
```

Output of the command history will resemble the following:

ID	Login user	Date and time	Action(s)	Altered
4	<student0>	2019-05-01 12:47	Install	1
3	<student0>	2019-05-01 11:51	Install	3
2	<qsecofr>	2019-04-30 10:27	Install	1
1	<student0>	2019-04-30 09:00	Install	9

history list

The 'yum history' command can also be used to undo actions taken by a particular yum command as well as to redo actions taken by a particular command.

Many other options/commands are available in yum. Do a google search on 'yum cheatsheet' to get more information.

NOTE: All of the package management functions can be accomplished via ACS as well as the command-line methods shown in this workshop.

Section 4. Introduction to the nano editor

There are many editors available within the open source space including the venerable 'vi' and 'nano'. If you are familiar with 'vi' and wish to use it for the rest of the lab by all means feel free to do so. This portion of the lab will introduce the nano editor and a few key commands that are needed to work with the editor.

__ 19. Enter the following command to enter your container:

```
chroot /QOpenSys/<team> /QOpenSys/usr/bin/bash
```

- o Where <team> is replaced with the team name from the token

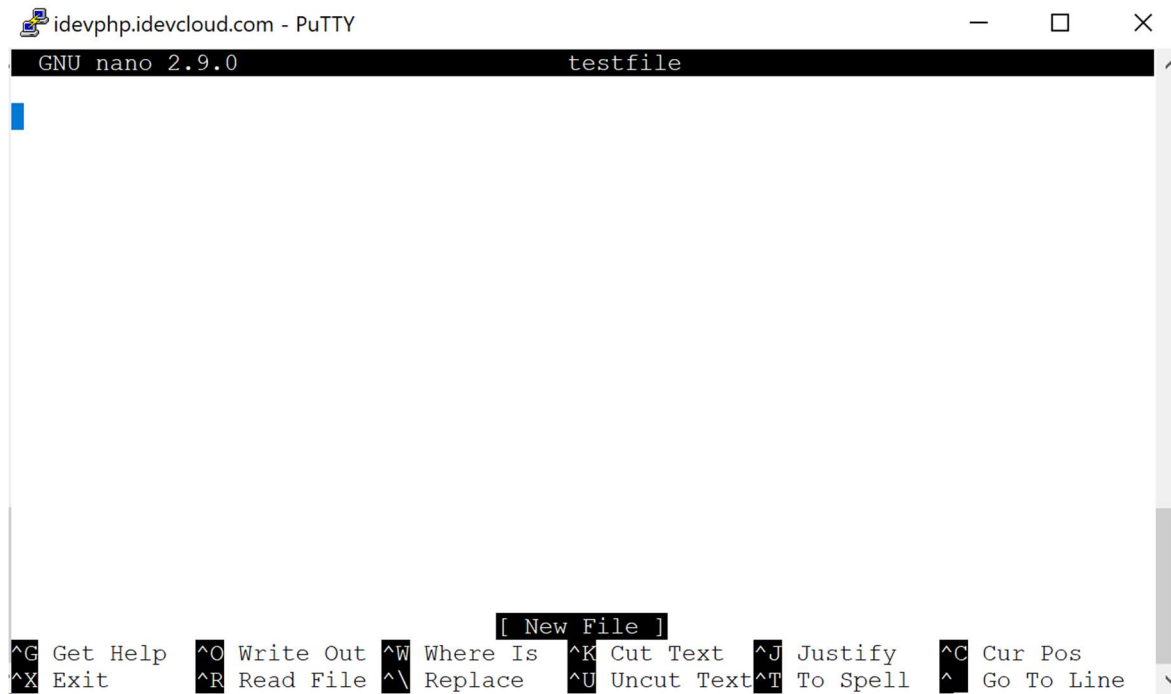
__ 20. Enter the following command to change to our home directory within your container (recall that we created this directory earlier in the lab):

```
cd
```

__ 21. Enter the following command to start the nano editor on a new file:

```
nano testfile
```

Notice that the nano editor is displayed:



```
idevphp.idevcloud.com - PuTTY
GNU nano 2.9.0 testfile

[ New File ]

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace  ^U Uncut Text ^T To Spell  ^_ Go To Line
```

Notice that at the bottom of the nano screen are a number of control-key sequences with functions such as Exit, Read File, and Write Out associated with them.

You might be tempted to use the mouse to position the cursor as well as highlight portions of text – don't! The editor knows nothing about the mouse, instead the arrow keys will be used to position the cursor.

- __ 22. Enter some text into the file
- __ 23. Press the <CTRL><X> key sequence to attempt to exit the editor
- __ 24. Note that a message is displayed at the bottom prompting for saving of the modified buffer. Press 'y' to save the file.
- __ 25. Notice that a new message prompting for confirmation of the name of the file to be written is displayed. Press <ENTER> to accept the file name which is the file name provided on the nano command in an earlier step.
- __ 26. Observe that the editor exits and the bash prompt is re-displayed.

Section 5. Exploring Python

One of the key areas of growth in the Open Source space on IBM i is the availability of additional open source programming languages. In this section we will take a brief look at Python.

This portion of the lab assumes that you are still in your container. If not issue the 'chroot' command as shown earlier in the lab.

- __ 27. This portion of the lab assumes that you are still in your container. If not then issue the following 'chroot' command to re-enter your container:

```
chroot /QOpenSys/<team> /QOpenSys/usr/bin/bash
```

- __ 28. Set the current working directory to the home directory and output it's path:

```
cd  
pwd
```

- __ 29. Create a file called `hello.py` with the following contents (keep in mind that Python uses spaces to indicate control blocks so ensure that you don't start any of these lines with extra spaces).

```
#  
# Hello World  
#  
print ("Hello Workshop Attendees!!!")
```

- __ 30. After saving the file execute it with the 'python3' command:

```
python3 hello.py
```

The following should be output:

```
Hello Workshop Attendees
```

Like the other open source languages, python has the ability to access Db2 resident data as well as ILE artifacts. The next portion of this exercise will walk through a sample script for accessing Db2.

__ 31. Use nano to create a script called `db2access.py` with the following content:

```
import ibm_db_dbi as db2

conn = db2.connect()

cur = conn.cursor()

cur.execute("SELECT CUST_ID,COMPANY, LASTNAME,CITY, STATE
FROM <TEAM>.SP_CUST WHERE COUNTRY='US'")

for row in cur:
    customerID,Company,LastName,City,State=row
    print(customerID,Company,State)
```

- Where <TEAM> is replaced with your team name from the token. This represents a library on the system.
- The `cur.execute` line is wrapped – this should be entered as a single line.

NOTE: Spacing is very important in python – it's how control blocks are defined. Ensure that the two lines in the for loop are lined up – typically use four (4) spaces

__ 32. Use the `python3` command to execute the script:

```
python3 db2access.py
```

Output of the script should resemble the following:

```
1221 Kauai Dive Shoppe      HI
1380 Blue Jack Aqua Center  HI
1510 Ocean Paradise         HI
1560 The Depth Charge       FL
1563 Blue Sports            OR
1624 Makai SCUBA Club       HI
```

Section 6. Exploring node.js

Another open source language that has become available to IBM i is node.js. With node.js javascript code can be executed on the server and as a result have access to server-side resources. The language is typically used to support web requests for various services. In this section of the workshop we will take a quick look at the language, implement a service and call the service from a browser. We will also take a look at how node.js can access Db2 data.

__ 33. Use nano to create a file called `hello.js` with the following content:

```
#!/usr/bin/env node
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(<port>, 'localhost');
console.log('Server running at http://localhost:<port>/');
```

- Where <port> is replaced with the value from your workshop token

__ 34. Now execute the script:

```
./hello.js
```

The console.log message should be output:

```
Server running at http://localhost:<port>/
```

NOTES:

- The script is able to be executed directly because of the first line in the file which is referred to as a 'shebang' – the `#!` is a special character sequence that when appearing on the first line of a script indicates the program/command to use to interpret the remaining lines of the script.
- The `./` designator when executing the command is a shortcut to the current working directory (or specifically the `.` is a short cut to the full path of the current directory). This is needed as the current directory is not in the execution search path.

At this point the script is waiting for requests on the specified port. Since 'localhost' was specified the script will need to be exercised from another terminal window:

- __ 35. Start another ssh session (via putty) to the workshop system using the values specified in the token
- __ 36. When logged in issue the following command to send a request to the node.js script (note, you don't need to enter your chroot container for this step):

```
curl http://localhost:10200/
```

The following should be output:

```
Hello World
```

This is the output that is specified as the 'res.end' in the script

- __ 37. Exit the second terminal window as it will not be needed again.
- __ 38. In the first terminal window (the one where the node.js script is running) press <CTRL><C> to terminate the program.
- __ 39. Let's change the script so that it will listen to requests on the public internet. Use nano to edit the file and change the two instances of 'localhost' to the <host> specified on the workshop token.
- __ 40. Start the node.js application:

```
./hello.js
```

The same console message should be output.

- __ 41. From a web browser attempt to execute the script by providing a URL of <http://<system>:<port>>

The hello World message should be output in the browser:



Up to this point executing the node.js script requires us to block a terminal widow while the process is running. The next step will take advantage of a node utility for background the script.

- ___ 42. Node Package Manager is used to install and maintain node.js based components. The following command installs a utility called pm2 (pm2 is a production process manager for Node.js applications – pm2 includes a built-in load balancer.)

```
npm install -g pm2
```

The installation will take a couple of minutes and output some diagnostic messages, the last of which should resemble the following:

```
+ pm2@3.5.0
added 310 packages from 258 contributors in 43.47s
```

Unlike yum|rpm which tends to install commands in the /QOpenSys/pkgs/bin directory which is included in the search path, npm tends to install its commands in the /QOpenSys/pkgs/lib/nodejs8/bin directory – that directory would either need to be added to the search path or specified when commands are executed.

- ___ 43. Issue the following command to use pm2 to execute the hello.js script:

```
/QOpenSys/pkgs/lib/nodejs8/bin/pm2 start hello.js
```

A number of diagnostic messages will be output followed by a table of node.js applets that are being managed by pm2:

Name	id	mode	status	↻	cpu	memory
hello	0	fork	online	0	0%	0 B

Like the other open source languages available on the platform node.js has the ability to access both Db2 as well as ILE programs and artifacts – this is accomplished through two additional modules that are available for the language:

- __ 44. To install the IBM Db2 for i access library execute the following command:

```
npm install idb-connector
```

A number of diagnostic messages will be output as the installation progresses. The end of the installation should have messages similar to the following:

```
+ idb-connector@1.1.10
added 68 packages from 68 contributors and audited
97 packages in 11.778s
found 0 vulnerabilities
```

- __ 45. To install the Node.js toolkit for IBM i execute the following command:

```
npm install toolkit
```

A number of diagnostic messages will be output as the installation progresses. The end of the installation should have messages similar to the following:

```
+ toolkit@1.5.4
```

added 1 package from 1 contributor and audited 99 packages in
3.68s
found 0 vulnerabilities

- __ 46. Use `nano` to create the following script. The script provides a simple example of using the `idb-connector` to access Db2 data from a `node.js` application:

```
const {dbconn, dbstmt} = require('idb-connector');

const sSQL = 'SELECT CITY FROM ZENDPHP7.SP_CUST';
const connection = new dbconn();
connection.conn('*LOCAL');
const statement = new dbstmt(connection);

statement.exec(sSQL, (x) => {
  console.log(JSON.stringify(x));
  statement.close();
  connection.disconnect();
  connection.close();
});
```

Unlike the hello world example, this script is not a web service, therefore we will want to execute the script directly in the terminal.

- __ 47. Use the `node` command to execute the script created in the previous step:

```
node <scriptname>
```

- Where `<scriptname>` is the name of the script created in the previous step

The script will output the value of city from each record in the table:

```
[{"CITY":"Kapaa Kauai    "}, {"CITY":"Freeport
  "}, {"CITY":"Kato Paphos    "}, {"CITY":"Grand Cayman
  "}, {"CITY":"Christiansted  "}, {"CITY":"Waipahu
```

```
"},{ "CITY": "Christiansted" }, { "CITY": "Kailua-Kona" }, { "CITY": "Bogota" }, { "CITY": "Kitchener" }, { "CITY": "Marathon" }, { "CITY": "Giribaldi"
```

The above is a truncated example of the output provided by the script. A couple items to make note of is that the output starts with an open bracket [and if the complete output had been shown it ends with a close bracket]. Additionally, each item retrieved from by the select statement is enclosed in braces {} and includes the field name as well as the field value. One could envision rather than outputting the data as is processing through this returned array and performing some level of additional processing on it.

This application could be turned into a web service by merging it with the web processing such as waiting for a request and providing a response shown in the previous example. Additionally, one could envision having a web form that would collect selection data from the user that could then be used in the select – the possibilities start to become endless.