

Enterprise-Wide Unique identifiers

Best practices for business continuity

Skill Level: Introductory

[Mr. Dana L. French \(dfrench@mtxia.com\)](mailto:dfrench@mtxia.com)

President
Mt Xia Inc.

27 Jan 2009

Successful implementation of business continuity in large IBM® AIX® environments is dependent upon the elimination of resource conflicts on the supporting computer systems. Many aspects of business continuity -- such as virtualization, high availability, and disaster recovery -- require unique identification values for many of the resource parameters involved in system configuration. Provided in this article is a discussion and mechanism for generating *Enterprise-Wide Unique* (EWU) identification values for a wide variety of configuration parameters.

Introduction

In a large IBM AIX computing environment spanning multiple data centers, the implementation of business continuity, disaster recovery, high availability, and virtualization will almost certainly be required. A necessity for designing and deploying large environments is to ensure that critical configuration parameters have Enterprise-Wide unique (EWU) identifiers. EWU identifiers are required to eliminate the occurrence of duplicate values that will interrupt and delay the processes of disaster recovery, high availability, and virtualization.

An EWU identifier is any parameter that has a single unique value across the entire enterprise. This identifier may be a character string, numeric, or alphanumeric value. The purpose of using EWU identifiers is to eliminate conflicting or duplicate values during all aspects of system administration. This includes normal operation, emergency, startup, shutdown, automated or manual fail-over, disaster recovery, or high availability procedures. The process of determining which parameters may

require EWU identifiers requires that an inventory of critical configuration parameters be assembled. Some of the items that are regarded as critical configuration parameters are:

- User names
- User ID numbers
- Group names
- Group ID numbers
- Cluster ID numbers
- Resource group names
- Volume group names
- Logical volume names
- File system mount point directory names
- Database record ID numbers
- Service group names
- HACMP application server names
- Application startup and shutdown scripts
- Error numbers

The previous list is not all inclusive, and is only provided here to show some examples.

Enterprise-Wide unique identifiers

One problem experienced by most systems administrators when attempting to implement EWU identifiers is how to track and document them. Most choose to create a database or spreadsheet to map and document the hundreds or thousands of identifiers that must be generated for all AIX systems in all locations and all data centers. This methodology is doomed to failure from the very beginning. Spreadsheets are often out-dated and are unreliable, and administrators can never be confident that a spreadsheet is current or even that it is the correct spreadsheet. Spreadsheets should never be used to store system configuration management information, but that discussion will be left for another article.

Databases have many of the same problems as spreadsheets with regard to storing system configuration management information, but usually there is only one database containing the information. So the systems administrators can at least be

confident that they are accessing data from the correct location, but they may or may not be confident that the information is up to date.

The best solution is to not use spreadsheets or databases for mapping EWU values to critical configuration parameters. The best solution is to dynamically generate the EWU value on an as-needed basis from the critical configuration parameter. This requires a mechanism that will generate the same EWU value across heterogeneous platforms. By generating the EWU value, the need for maintaining spreadsheets and databases is eliminated. The most portable mechanism across UNIX® platforms for generating the EWU value is a shell script.

If a shell script is used to generate the EWU value, then it must be able to accept any given input string of characters. The output value must be a user selectable format and number of digits. The reason for using a shell script is to be able to consistently generate the same EWU identifier value for any given string on any UNIX system. For example, whenever you need to know the UID number for a user name, instead of having to look it up on a spreadsheet or in a database, you can generate it on any UNIX system using the EWU value-generator script.

A script for generating the EWU value should be written as a function and included in an organization's standard set of shell functions. This enables systems administrators to easily incorporate this function into their shell scripts and ensure EWU values are being generated in a consistent and repeatable manner. The script function should be able to generate EWU values in multiple formats, such as base 10, binary, octal, or hexadecimal. The number of output digits must also be selectable in order to accommodate the needs and requirements of a variety of purposes and applications.

- **base:**
 - base 16 - hexadecimal (0-f)
 - base 10 - normal counting digits 0-9
 - base 8 - octal (0-7)
 - base 2 - binary (0-1)

Here is an example of the EWU value calculation using AIX commands and syntax:

```
print "any string requiring an enterprise wide unique identifier" | cksum | awk '{ print $1
}'
341023782
```

Using the previously generated CRC cksum value, calculate the EWU value:

d	Number of digits in output value
---	----------------------------------

b	counting base (decimal, hexadecimal, octal, binary)
c	cksum value
m	modulo from dividing the CRC cksum value by the counting base raised to the number of output digits

base 10 (decimal):

```
d=4
c = 341023782
m = c % 10**d
m = 3782
```

base 16 (Hexadecimal):

```
d = 4
m = c % 16**d
m = 39976 = 9C26 HEXADECIMAL
```

base 8 (Octal):

```
d = 4
m = c % 8**d
m = 3110 = 6046 OCTAL
```

base 2 (Binary):

```
d = 4
m = c % 2**d
m = 6 = 0110 BINARY
```

Here is a base 16 (hexadecimal) EWU value calculation in Korn Shell 93 syntax:

```
#!/usr/bin/ksh93
d=4
b=16
v="any string requiring an enterprise wide unique identifier"
c=$( print -- ${v} | cksum | awk '{ print $1 }' )
(( m = c % b**d ))
print -- "ibase=10; obase=${b}; ${m}" | bc
9C26
```

Here is a base 10 (decimal) EWU value calculation in Korn Shell 93 syntax:

```
#!/usr/bin/ksh93
d=4
b=10
v="any string requiring an enterprise wide unique identifier"
c=$( print -- ${v} | cksum | awk '{ print $1 }' )
(( m = c % b**d ))
print -- "ibase=10; obase=${b}; ${m}" | bc
3782
```

Here is a base 8 (octal) EWU value calculation in Korn Shell 93 syntax:

```
#!/usr/bin/ksh93
d=4
b=8
v="any string requiring an enterprise wide unique identifier"
c=$( print -- ${v} | cksum | awk '{ print $1 }' )
(( m = c % b**d ))
print -- "ibase=10; obase=${b}; ${m}" | bc
6046
```

Here is a base 2 (binary) EWU value calculation in Korn Shell 93 syntax:

```
#!/usr/bin/ksh93
d=4
b=2
v="any string requiring an enterprise wide unique identifier"
c=$( print -- ${v} | cksum | awk '{ print $1 }' )
(( m = c % b**d ))
print -- "ibase=10; obase=${b}; ${m}" | bc
110
```

Here is an explanation of the components of the EWU value calculation using the CRC cksum command:

- **cksum:**
A CRC cksum is calculated for each string provided by the user on the command line.
- **denominator:**
This is calculated from the counting base and the number of digits desired in the output value. This value is also the number of unique possibilities for the output values using the counting base and number of digits in the output value.
- **modulo:**
The remainder from dividing the cksum value by the denom value. This is the base 10 unique value for the string provided by the user on the command line.

- **ewuid:**
The EWU value translated into the desired output counting base and number of output characters.

The previous technique will generate a unique identifier for any user-supplied character string; however, it is somewhat cumbersome to issue this list of commands every time an EWU value is needed. Fortunately, a script already exists that uses this technique and can be used to generate these EWU identifiers in a wide variety of counting bases while permitting the user to select the desired number of output digits. This script is called `mkewuid` and is available [here](#).

The following usage message information is from the `mkewuid` Korn Shell script function:

```
Generate an Enterprise Wide Unique (EWU) identifier
value for any given input string of characters. For
example, this script can be used to generate an
Enterprise Wide Unique UID number for any given user
name. The output value may be calculated in a user
selectable format and number of digits.

Usage: ${1##*/} [-?vV] [-hdoB] [-mM] [-c] [-p #] [-ul] [string...]
Where:
  -h = Hexadecimal (Base 16) output value (default)
  -d = Base 10 output value
  -o = Octal (Base 8) output value
  -b = Binary (Base 2) output value
  -p # = Number of characters in output value (default: 4)
  -c = Display column headers in output (default: no headers)
  -m = Minimize output
  -M = Maximize output (default)
  -u = Uppercase Enterprise Wide Unique output values
  -l = Lowercase Enterprise Wide Unique output values (default)>
  -v = Verbose mode - displays mkewuid function info
  -V = Very Verbose Mode - debug output displayed
  -? = Help - display this message
  string = any alphanumeric string such as user names, group names, host names, node
names, etc. (default: local node name )

Example Usage:
mkewuid -c -h -p 6 teststring1

Example Output:
# string : base : cksum : denom: modulo : ewuid
teststring1:16:542276492:16777216:5405580:527b8c
```

The `mkewuid` script function will output a record line for each string of characters provided on the command line.

To accommodate variable requirements by applications and databases, the `mkewuid` script function can convert the output values to all upper- or lower-case characters based upon the command-line options `-u` or `-l`, respectively.

The default output record from the "mkewuid" script function consists of several

fields of information delimited by a colon (:) character:

- **Field 1:** The user-provided string of characters on the command line.
- **Field 2:** The counting base used to calculate the EWU value.
- **Field 3:** The CRC cksum value of the user-provided string of characters.
- **Field 4:** The denominator calculated from the counting base and the desired number of digits in the EWU output value.
- **Field 5:** The EWU output value.

A command-line option is provided to minimize the output record to only display the EWU value for each character string provided on the command line. The minimize option is **-m**; however, the default behavior of the script function is to maximize (**-M**) the output record data and display all fields. In normal usage, this script function would normally be called from a shell script written by a systems administrator, and used to calculate an EWU value for a single character string. The output would be captured into a shell variable and used by subsequent commands in the shell script that called the "mkewuid" function:

```
#!/usr/bin/ksh93
...
...
...
EWUID=$( mkewuid -d -p 7 -m myusername )
...
...
...
```

The example above illustrates how the mkewuid script function may be used to dynamically calculate the UID number of a user name. And since this script will always generate the same number for the same character string on any system, the need for storing the UID number in a spreadsheet or database is eliminated.

Conclusions

The reasons for using a script mechanism, such as mkewuid, to generate EWU values are numerous and important for the operation and management of a modern data center environment. Systems administrators must have a consistent, repeatable, and portable method of ensuring unique values for the purpose of identifying various hardware, software, facilities, network, and personnel resources. The problem of duplicate identification values for critical resources can cause major problems during the implementation and operation of business continuity principles. Disaster recovery and high availability fail-overs require that resources involved in these processes be uniquely identified across all nodes. Resource identification conflicts during fail-overs, whether automated or manual, can result in extended

downtime while the conflicts are resolved. Designing these resources with EWU values eliminates the possibility of conflict during automated or manual fail-overs, high-availability operations, and disaster-recovery testing or implementation.

Another benefit of using a standardized mechanism of generating EWU values is that it eliminates the need for storing these values in spreadsheets or databases. Spreadsheets are notoriously unreliable for the storage of this type of data. Invariably an administrator will make a copy of the spreadsheet, make changes to that copy, then days later copy it back to the central location. In the time between, another administrator has made changes to the spreadsheet and these changes are overwritten. Or, more commonly, the spreadsheet is simply not be updated on a timely basis and is consistently out of date. The same is true with using databases for the storage of information. The data is typically out of date, or when the data is needed in an emergency situation, the database is inaccessible. For these reasons, spreadsheets and databases must be avoided for the storage of critical configuration information. When possible, configuration information should be dynamically generated in a consistent and repeatable manner.

Although compiled programs are generally much faster than scripts, speed is not the primary concern when creating a mechanism for generating EWU values. The primary concern is to create a platform-independent mechanism that can be copied between systems with little effort. A compiled program will need to be recompiled to move it to another platform, and thus will require a compiler. Although these binaries can be generated for each platform, it requires an additional level of expertise to accomplish this task. This level of expertise may not be available during a disaster recovery implementation. The best mechanism for this purpose is to use a shell script.

Finally, the most important reason for dynamically generating EWU values is data center automation. When implementing data center automation processes, it is imperative that consistent, repeatable, and reliable mechanisms be used to support this structure. This will immediately rule out the use of spreadsheets because they will be unreliable and inconsistent for this purpose. Databases provide reliability, but that assumes the deployment processes have access to the network on which the databases exist during the deployment process, which may or may not be true. Also during a disaster-recovery implementation, automated deployment will likely be required long before the databases are restored. Under this scenario the availability of critical configuration information for the data center automation processes will be unlikely. Again, the best solution is to dynamically generate critical configuration information when possible.

Resources

Learn

- [Enterprise Wide Unique identifier generator](#) generates an Enterprise Wide Unique UID number for any given user name.
- [AIX Disaster Recovery: Resolving Resource Conflicts](#)(developerWorks, September 2007) identifies resource conflicts that typically occur during a disaster recovery implementation and provides suggestions for resolving these conflicts.
- [Korn Shell Scripting Template](#) is a korn shell scripting template for creating new korn shell scripts.
- [The AIX and UNIX developerWorks zone](#) provides a wealth of information relating to all aspects of IBM AIX systems administration and expanding your UNIX skills.
- [New to AIX and UNIX?](#) Visit the New to AIX and UNIX page to learn more.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.
- [AIX](#): Visit this collaborative environment for technical information related to AIX.
- [Podcasts](#): Tune in and catch up with IBM technical experts.

Get products and technologies

- [IBM trial software](#): Build your next development project with software for download directly from developerWorks.

Discuss

- Participate in the AIX and UNIX forums:
 - [AIX Forum](#)
 - [AIX Forum for developers](#)
 - [Cluster Systems Management](#)
 - [IBM Support Assistant forum](#)
 - [Performance Tools forum](#)
 - [Virtualization Forum](#)
 - [More AIX and UNIX forums](#)
 - [AIX networking](#)

About the author

Mr. Dana L. French

Mr. French's career in the IT industry has spanned three decades and numerous industries. His work focuses primarily on the fields of business continuity, disaster recovery, and high availability, and he has designed and written numerous software packages to automate the processes of business continuity, disaster recovery and high availability. He is most noted for his approach to system administration as an automated, business oriented process, rather than as a system oriented, interactive process. He is also a noted authority on the subject of Korn Shell programming.

Trademarks

IBM and AIX are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.