Damage Detection Tool

Jun. 15th, 2014

Tim Rowe

Damage Detection Tool

For many users when a damaged object is encountered it is often at the most inconvenient time possible. Often a damaged object is encountered during something like a system save. When that happens, it pretty much makes a mess of any save window you had. When a damaged object is encountered during a save, the save process is stopped at that point and can not proceed until that damaged object is cleaned up. Then you have to restart the save. We realize this is causes much difficulty, so we have created a new tool that can help you proactively find some of these objects that may be damaged. Of course our goal is that we don't ever have damaged objects, but on occasion there are situations that occur that will cause an object to become damaged. When that happens, this new tool can be leveraged to help you find these on a schedule that meets your time.

The new damage detection tool is based on the Application Runtime Expert (ARE) framework. The ARE tool is designed to help you better understand what has changed on your systems and the environments that your applications run in. This tool has 2 components, the core runtime engine which is a part of the base operating system. The Core provides the necessary framework and runtime engine to allow you to run a verification on any system and return the results in a easy to review report. The 2nd part or ARE is the actual product itself 5733-ARE. The ARE product is the GUI interface that allows you to build custom collections of attributes or system content that you want to keep track of. It also contains a runtime interface where you can run in a real time manner or as a scheduled comparison that can be run on a regular basis.

What does the new tool do

The new damage detection tool can be run on any system that has the correct level of PTF applied. The damage detection tool has been built to find objects with in the systems disks that contain data checks. The tool we not actually search though system objects, but rather actually reads every sector on the specified disks looking for a data section that contains a data check. A data check is when the actual data write fails for some reason. There are several reasons that a write to disk fails. Once a disk sector has a data check in it, that will cause the object that sector belongs to be reported as damaged the next time that area is accessed.

The damage detection tool can be used to check every disk on the entire system, a specific iASP, a specific disk on the system or even a region within a disk. The tool will scan every sector on the disk looking for malformed data. Once a data check is encountered, that disk sector is reported by the damage detection tool. The tool will then attempted to determine what actual ILE object that sector is associated with. This is often a difficult process and is not always successful depending on where the actual damage is within that object.

Here is a simplistic example of how sectors are stored on a disk and how ILE objects are made up. You can see that OBJ1 is made up of 3 different pages. But OBJ1 is damaged since the link that connects Page1 to Page 2 is broken.

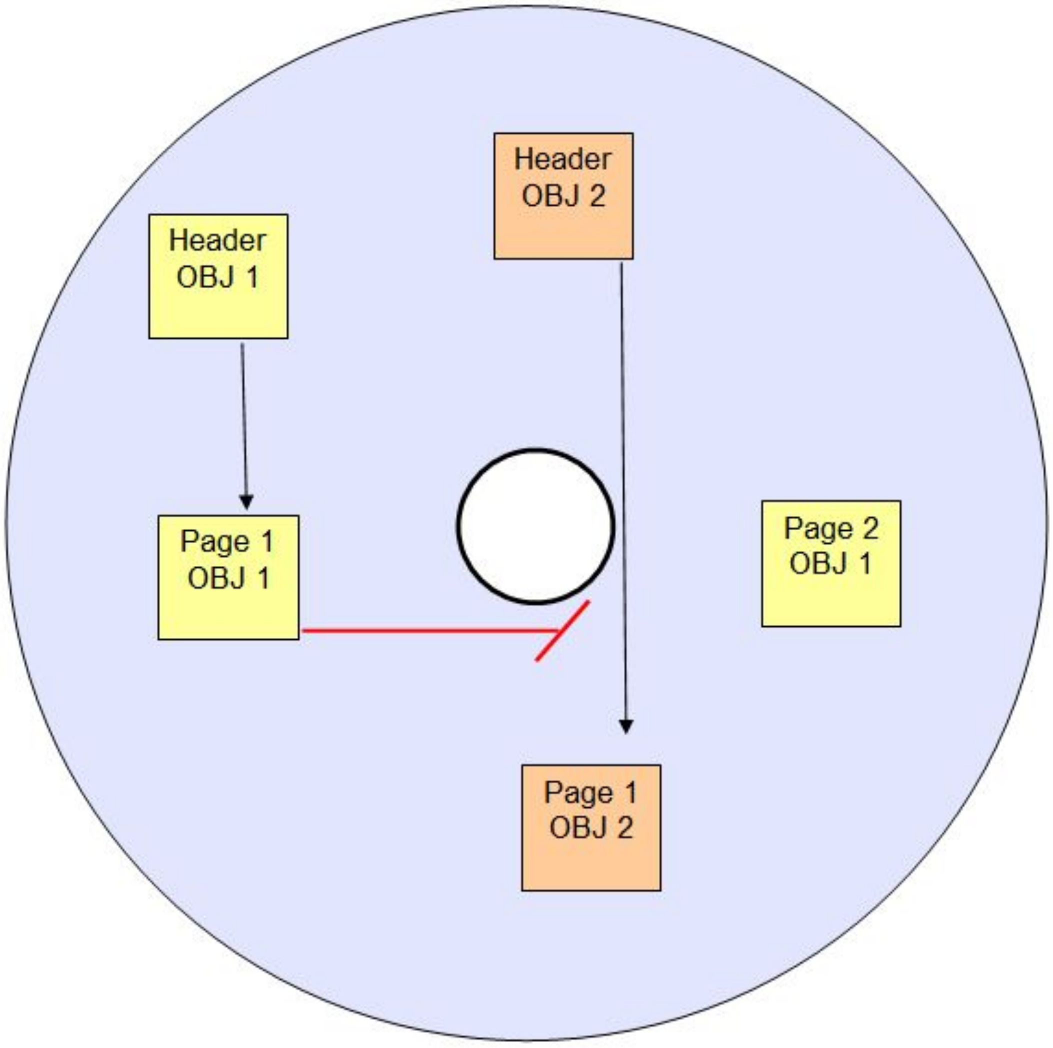
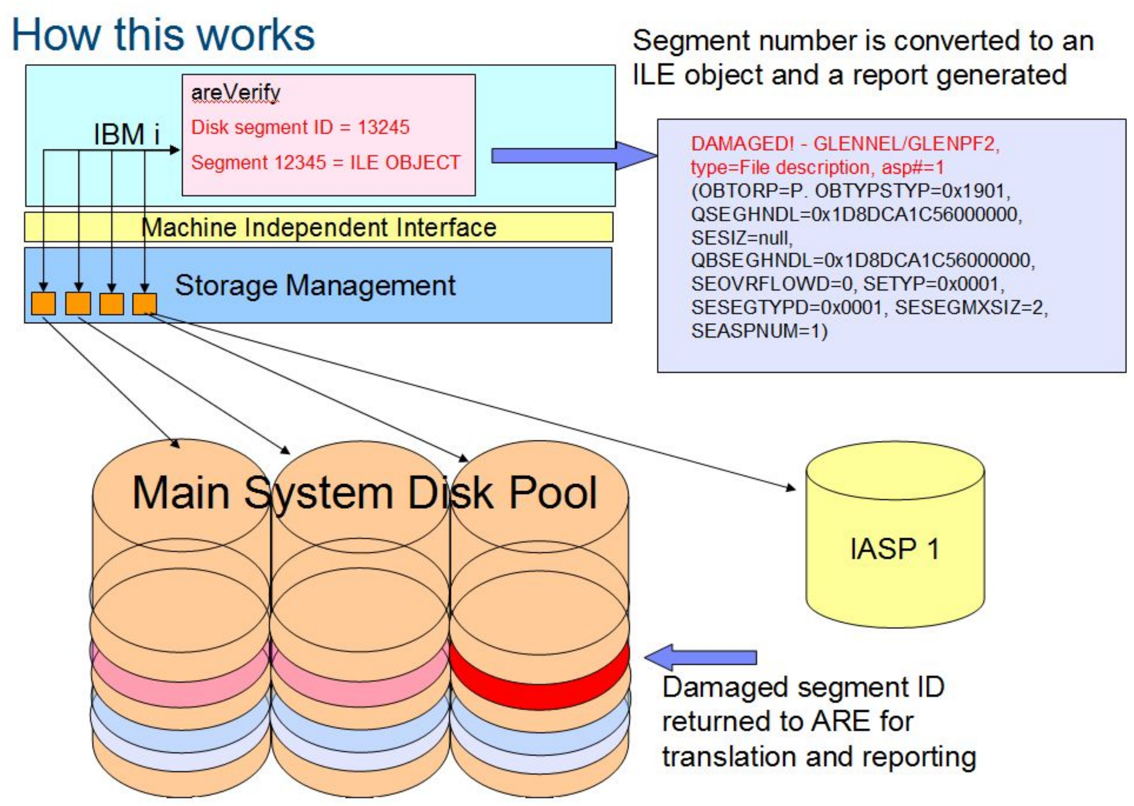


Figure 1 - Example of damaged ILE object

When the damage detection tools finds that data check it will report both the actual disk sector and that ILE object that it represents. The tool does the checking in a multi-threaded manner. There are controls that give you the ability to control how fast or slow the scan is done.


How this works
Segment number is converted to an ILE object and a report generated
areVerify
Disk segment ID = 13245
Segment 12345 = ILE OBJECT
IBM i
Machine Independent Interface
Storage Management
Main System Disk Pool
IASP 1
Damaged segment ID returned to ARE for translation and reporting

```
DAMAGED! - GLENNEL/GLENPF2,
type=File description, asp#=1
(OBTORP=P. OBTYPSTYP=0x1901,
QSEGHNDL=0x1D8DCA1C56000000,
SESIZ=null,
QBSEGHNDL=0x1D8DCA1C56000000,
SEOVRFLOWD=0, SETYP=0x0001,
SESEGTYPD=0x0001, SESEGMXSIZ=2,
SEASPNUM=1)
```

Figure 2 - example of a full system scan and a damaged object reported

The above figure details an example of how the tool works. Showing the how the tool starts up many threads. Each scanning a section of the disks looking for issues. When a data check is encountered it is reported to the ARE infrastructure and the associated ILE object is resolved and recorded in the report.

Details on how to run the tool

In order to run this tool you must be signed on with a user profile that is either QSECOFR or has *ALLOBJ special authority. This tool is actually a script that is run in the Qshell environment. The actual process for the tool runs in two separate phases. The first phase is a dump of the directory. This is the process that will build the list of disk segments that will then actually be scanned in the second phase. Both of these phases can be controlled with parameters. You can choose to skip the director dump you can specify the number of processes. The second phase is run at the lowest levels, and you can specify the number of threads needed to scan all the disks in a manner that will ensure minimal impact to the running of the system.

Enter the following to run this tool:
From the command line - qsh

Here are a couple of examples:

- Check all storage units (sys base) (Qshell command):
  - */QIBM/ProdData/OS/OSGi/templates/bin/areVerify.sh -storage diskUnits=*ALL*
- Check certain storage units (Qshell command):
  - */QIBM/ProdData/OS/OSGi/templates/bin/areVerify.sh -storage diskUnits=1,2,4*
- Check specified segment
  - */QIBM/ProdData/OS/OSGi/templates/bin/areVerify.sh -storage checkSegment=0x1122334455660000*

All Qshell scripts can also be run directly from the command line by submitting a job from the CL command line

- Check all disk units (sys base), in background (**CL command**):
  - *SBMJOB CMD(STRQSH CMD('/QIBM/ProdData/OS/OSGi/templates/bin/areVerify.sh -storage confirm=true diskUnits=*ALL'))*

There are a number of additional parameters that control a number of additional functions and features. They are all described below.

Damage Detection tool parameter specifications

Command syntax:

- /QIBM/ProdData/OS/OSGi/templates/bin/areVerify.sh -storage [<parameter_key=value>[,<parameter_key=value>]...]

Mandatory parameters – One of the following parameters must be specified:

- **diskUnits**=<comma separated numbers>
  - Description: Specifies which disk units to check.
  - Default value: N/A.
  - Example: *diskUnits=1,2,4*   *diskUnits=*ALL*

- **checkSegment**=<hexValue>
  - Description: Check a specific segment by its address
  - Default value: N/A.
  - Example: *checkSegment=0x1122334455660000*

Optional parameters for controlling the Directory Dump phase – only specify if you want to override the default behavior.

- **skipDirDump**=<bool>
  - Description: if true is specified, the directory dump phase will be skipped. If you have run the tool before and there's no changes in storage you care about, use this parameter to skip the directory dump phase, which takes long time.
  - Default value: false
  - Example: *skipDirDump=true*

- **dbName**=<temp DB name for dirdump data>
  - Description: Specifies the temporary library name to store the directory dump data. This parameter is only for directory dump phase.
  - Default value: QTMPAREDDD
  - Example: *dbName=MYLIB*

- **dirType**=[P|I|T]
  - Description: Identifies the directories for which data should be collected. This parameter is only for directory dump phase. The possible values for this parameter are:
    - T: Temporary
    - P: Sysbase permanent & user ASPs
    - I: Independent ASP
  - Default value: P
  - Example: *dirType=I*

- **iASP**=<number>
  - Description: identifies the IASP number if 'I' was selected for the "Directory identifier" parameter. If a value other than 'I' was selected for the "Directory identifier" parameter this parameter is ignored. This parameter is only for directory dump phase.
  - Default value: 0
  - Example: *iASP=3*

- **jobQueue**=<name>
  - Description: identifies the name of the job queue which will be used for the background jobs which collect the requested data. This parameter is only for directory dump phase.
  - Default value: QCTL

- **jobQueueLib**=<name>
  - Description: identifies the library of the job queue which will be used for the background jobs which collect the requested data. This parameter is only for directory dump phase.
  - Default value: QSYS

- **jobCount**=<number>
  - Description: identifies the number of background jobs which will be used to collect the requested data. This value must be a number between 5 and 100. This parameter is only for directory dump phase.
  - Default value: 30

Optional parameters for the disk scan phase – only needed if you want to over ride the defaults.

- **skipPageVerification**=<bool>
  - Description: specifies whether the Page Verification phase will be performed.
  - Default value: true

- **threadCount**=<number>
  - Description: Specifies the thread count for Page Verification. This value must be a number between 1 and 100.

Optional general usage parameters – only needed if you want to override the defaults

- **op**=[check | clear]
  - Description: Operation mode, used only when parameter *diskUnits* is specified.
    - *check*: check segment error in specified disk units.
    - *clear*: clear error flags in free space of the specified disk units. When this option is specified, only the first disk unit will be processed, and other disk units are ignored.
  - Default value: check

- **statusUpdateInterval**=<number>: minutes of status update
  - Description: Specifies the status update interval, in minutes. The status message will be written to console, and/or log file. This value must be a number between 1 and 1440.
  - Default value: 10

- **outputFile**=<IFS file name>
  - Description: Specifies the log file
  - Default value: /tmp/areDodReport.txt

- **confirm**=<boolean>
  - Description: Controls whether user confirmation is required before starting the task. If true, no user confirmation prompt is shown.
  - Default value: false

- **version**
  - Description: Show version of the tool

How to get the tool

The damage detection tool has been made available via a number of PTFs. Once they are all applied you can start using this tool,

System Dependencies

- IBM i 7.1
  - 5770SS1 option 3 – Extended Base Directory Support
  - 5770SS1 option 30 – QShell
  - 5770SS1 option 33 – PASE
  - 5761JV1 option 8 or 11 – J2SE (5 or 6) 32 bit
- IBM i 6.1
  - 5760SS1 option 3 – Extended Base Directory Support
  - 5760SS1 option 30 – QShell
  - 5760SS1 option 33 – PASE
  - 5761JV1 option 8 or 11 – J2SE (5 or 6) 32 bit

Required PTFs

- V7R1
  - **SI50374** (5770SS1)
  - Make sure the following dist-req are applied:
    - Req: MF56898
    - Req: MF56876
    - Req: SI45469
- V7R1
  - **SI45499** (5761SS1)
  - Make sure the following dist-req are applied:
    - SI51025 (5761SS1)
    - SI30796 (5761SS1)
    - R610:
      - MF57435 (5761999)
      - MF57436 (5761999)
    - R611:
      - MF57425 (5761999)
      - MF57426 (5761999)

Known issues – There are a couple of items that need to be taken under consideration.

- If any JDBC driver is not found or fail to be created, update JDBC by these PTFs:
  - IBM i 7.1: JDBC Driver: SI50669 (5770SS1)
  - IBM i 6.1: JDBC Driver: SI49252 (5761SS1)

- CCSID setting

  - *Job CCSID must NOT be 65535.*
  - *The CCSID for the user profile used on the job that contains the JVM must NOT be 65535. Use the* ***CHGUSRPRF*** *command to update the user profile or, if the profile is configured for *SYSVAL, consider updating the QCCSID system value. It is also recommended that the user profile used on the JDBC connection be configured with the same CCSID. Refer to the IBM® iSeries™ Information Center topic* ***Language identifiers and associated default CCSIDs*** *for further information on selecting the appropriate CCSID.*

Potential impacts to the system

- The tool has the potential to cause heavy IO when running

- The tool (Directory Dump phase) creates a temporary library
  - Size is less than 5/1000 of total storage used
  - The temp library is not deleted automatically.
    - By default the library is QTMPAREDDD

- System impacts can be managed with the following parms:
  - jobCount: [5, 100], for Directory Dump phase
  - threadCount: [1, 100], for Page Verification phase
  - skipDirDump: set to true to skip the directory dump phase, if:
    - Previously directory dump has been performed, and
    - No useful storage update since the last dump, and
    - The previously generated temp library (by default QTMPAREDDD) is not deleted/changed.

Our experience with the tool

We have run this tool on a large number of systems here within the lab in Rochester and we have also had the tool run on a number of end user systems. The time it take to actually run the verification on a system is dependent on the size of the disk pool being scanned and the number of resources that are allocated to the running of this tool Here are some details on the impacts to the system and the time it took to run.

- Example 1 – Large system
  - 7TB of internal storage
  - Approx 58% used
  - 3 hour 18 min to complete full disk scans.
- Example 2 – Small system
  - 352GB internal storage
  - 15% used

    - 4 min to complete scan

As you can see in both of these cases, the length of time to complete the run was reasonable. In both cases, a production workload was running an there was little to no impact to the running workload.