Fixing Status 146 Problems with Actian PSQL/Zen

A White Paper From

For more information, see our web site at
http://www.goldstarsoftware.com

Fixing Status 146 Problems with Actian PSQL/Zen

Last Updated: July 2022

This paper discusses the Status 146 message that can be returned from an Actian PSQL/Zen database environment in some depth. This is a fairly complicated issue, so the description is complicated as well.

What is a Status 146?

Status 146 is defined by Actian in the Zen v15.10 documentation as:

> ***146: Duplicate system key:*** *The same key number was generated by two different threads generating system keys.*

However, even this text description is not entirely accurate, as it actually can be returned when ANY duplication of the system data values is seen – not just from two threads at the same time.

To understand this error, we need to first review the concept of **System Data**.

What is System Data?

System Data was first introduced with the Pervasive.SQL 7 database engine and the v7 file format. System Data is a special, hidden, 8-byte data block that can be optionally stored with database records that provides a guaranteed unique value that can be used for transaction tracking purposes, including the ability to roll changes forward after an engine failure. It is accompanied by a hidden data file key (Unique Key Number 125) that allows it to be used to directly look up any record by its SysData value.

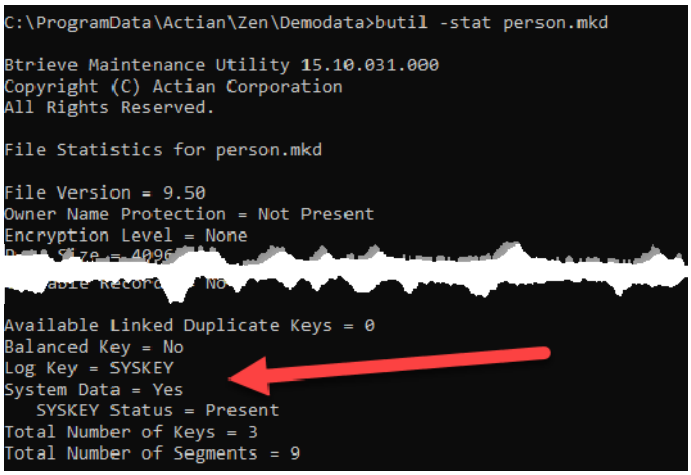
System Data was later used as a unique identifier for records from other environments, including Actian's DataExchange and Goldstar Software's GSSync replication solutions, since it offered a way to uniquely identify ANY record with a static value that would not change for the life of the record, making it suitable as a tracking value for such solutions.

According to the original PSQL7 documentation, the system data value was supposed to be a Timestamp value that indicated the time at which the record was inserted into the system. However, this was eventually removed from the documentation when it was found that the value was off by a month – it used the OS call that counted January as 0 instead of 1 – and therefore it was not a true PSQL Timestamp value. From that point forward, it was simply called "an 8-byte binary value, guaranteed to be unique in the file."

[N.b. In the v15 database engine, the v13 file format was extended with a new feature called SysData2, which changes this data value to a TRUE septa-second Timestamp field, eliminating this extra issue. However, that is not really germane to this discussion.]

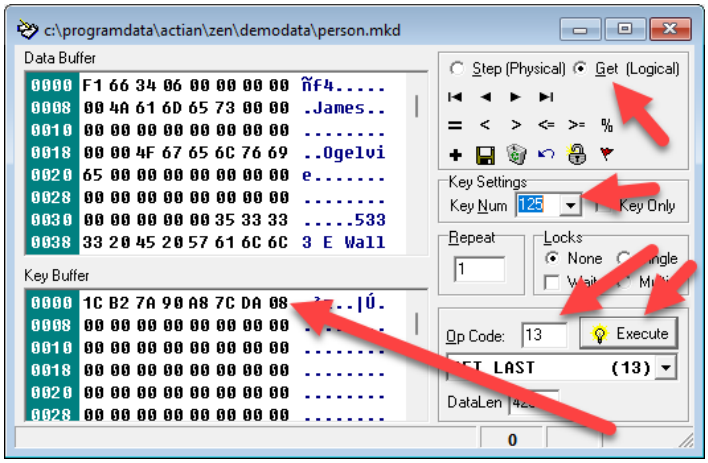
Can I See System Data Values in an Existing File?

Although the System Data is invisible to the typical application, it is still accessible from the Btrieve interface (and from SQL, in v15). To find out if you have System Data in your database files, do a simple BUTIL -STAT operation on the file and look for this block of text:

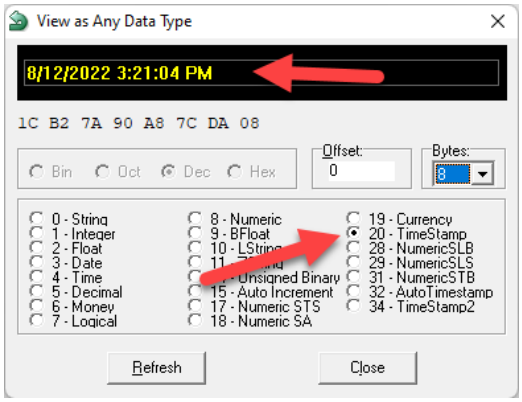


This text indicates that the System Data block is present in the file, and that the System Data Key (Key Number 125) is also available. Further, it confirms that the SysKey is being used for the transaction logging on this file.

Further, you can see the System Data values by opening up the database file within the Function Executor. Then ,select the Get (Logical) radio button, select Key Num 125, and the issue a GetFirst or GetLast operation:



The Key Buffer will now show you the System Data value for that record. As this is a 64-bit value in little-endian format, you read it backwards, so it is 0x08BA7CA8907AB21C, which is a decimal value 628,952,161,391,915,548. If you want, you can right-click on the first byte and select Show As Timestamp to see the value interpretted as a timestamp:



In this sample file, the data record was actually generated on 2022-07-13, but due to the off-by-one error on the month (and the fact that there are 31 days in July), it gets displayed as 2022-08-12 in this utility instead.

What is the LastInsertTimestamp?

To ensure that the System Data values are always increasing, and never experience any duplicate values, the engine keeps track of the latest time used for every file in the environment. It does this by storing a value within each file's header page (the File Control Record, or FCR), called the *LastInsertTimestamp*. This value represents the timestamp (or System Data value) in use when the last insert operation was attempted on that file. The engine maintains this value for all files, even those not using System Data, because the System Data key can always be added later on.

Can I See the LastInsertTimestamp in an Existing File?

The LastInsertTimestamp value is an internal value which typically has no importance outside of the database environment, to the developer, or to the end user. As such, Actian does not offer a way to view the LastInsertTimestamp value from the FCR from their own tools.

However, the Goldstar Software tool GSRecover can expose this value for you and provide a window into the data values used therein.

What is Synthetic Time?

As mentioned above, the System Data value should always be increasing, and it must be guaranteed to be unique. However, there are times when "clock time" moves backwards, such as in the Daylight Savings Time fall-back, or if the clock on the server was set incorrectly and suddenly gets adjusted. Such a change can cause a specific time to be used twice, and therefore create a duplicate System Data value, which would not be good.

The Actian database engine is smart enough to handle this possibility. The engine is able to see the current time (as reported by the OS), but it also tracks the *highest* time value recently seen in any file. If it detects that the time has moved backwards, it changes from using the real time to a *synthetic time* value. This synthetic time is then used as a simple counter, adding one with each insert operation. By jumping to this linear counter, this special synthetic time computation can allow the engine to survive periods of time where the clock moves backwards without generating any duplicate System Data

values, and maintaining uniqueness among these values as well. Once the real time surpasses the synthetic time, the real time can be used again.

What Happens When Synthetic Time Goes Bad?

In certain situations which have yet to be fully understood, something can happen to the operating system, database engine, real-time clock, or some other component that causes an artificially-high time value to be seen by the engine. If the engine gets a strange return value that is LESS than the current time, then it uses the current time and it uses real time for the System Data values as expected. However, if some OS call returns a value that is HIGHER than the current time, the engine can flip to using synthetic time (based on that value) from that point forward.

As previously noted, using synthetic time isn't normally an issue, as it will eventually get caught up. However, in some cases, that value is substantially higher – like hundreds or thousands of years higher – and this can force the engine to return synthetic time from that point forward. This synthetic time value is then used as the System Data value when needed in any subsequent file operations. However, due to the mechanism behind ensuring that time never goes backwards, the LastInsertTimestamp value in each file touched ALSO gets updated to the new value.

Under normal operations, it shouldn't matter whether you are using real or synthetic time – the system operates and performs equally well. However, when a synthetic time with a large difference is used, then things can get a bit dicey when the engine is restarted. When the engine first starts, it will always use the current "real time" value for all operations. However, once an insert is attempted on a file, it will update its internal counter to the maximum of either the current time or the LastInsertedTimestamp value +1 of the file being inserted into. The sudden "jump" in the values can cause some System Data values to be "normal" timestamp values, and some to be synthetic times. Again, this is not a problem normally, as System Data values are still always increasing (even with the big jump), and this works perfectly.

However, there are a few cases where we have seen issues reported:

1. **Wrap-Around Values**: We have seen cases where a very large value gets picked for a synthetic time, with one file seeing its first System Data value of 0xFFFFFFFFFFFFFFF8. Once 8 records have been inserted into this file, it wraps around to 0 reports a Status 146.
2. **Out-of-Order System Data Values**: We have seen cases where a file is rebuilt after an engine restart and all normal SysData values are used. Then, a second file is accessed with a LastInsertTimestamp using a synthetic time value causes the SysData value to jump. A subsequent insert into the first file will create a record with the synthetic value. If you restart the engine once again, rebuild the file (retaining System Data values with the -sK option), and attempt another insert, the engine will use normal time again, and you will be inserting records with System Data values that are LOWER than the highest System Data value in the file, resulting in a System Data that jumps backwards in time.
3. **Duplicate System Data Values**: Similar to the above, it is possible to flip a data file to synthetic time after accessing a *second* data file, which then starts using that value as the base for new System Data values. If you then restart the engine and rebuild the first file with the -sK option, and then attempt an insert on the second file again, the internal synthetic time value will be the same value that was already inserted. When you attempt the next insert on the first file yet again, you'll find that this value is already in use, and get a Status 146 back.

As a result, once these values go bad, you need to take some steps to fix the problem and hopefully prevent it from recurring.

Fixing a Status 146 in a Data File

Fixing a Status 146 problem in an environment may be easy, or it may be difficult. It could be as simple as restarting the database engine. Or, it could require restarting the engine and rebuilding the files in question to use new System Data values. Or, if you are not reliant on the System Data values for anything (i.e. not using replication) and the files already have a unique key, then you might be able to simply drop the SYSKEY index from the file entirely.

Do I Need to Fix a Synthetic Time?

If you have read this far, then you ALSO realize by now that these fixes are more of a band-aid fix to address the specific issue being seen, without doing anything about the root cause, which is the synthetic time. If the system is working on synthetic time, then there must be one or more files that have flipped the engine to that synthetic time, and even if you fix the file reporting the Status 146, the engine always has the chance to flip back to the same synthetic time value again in the future, and the problem can recur. In short, you may be fighting this for a long time to come.

The better question is whether you should even care about the synthetic time at all. As long as the synthetic time value is not inordinately huge (as in the example 0xFFFFFFFFFFFFFFF8 above), then it should not present any issues. Even a seemingly large value like 0xFFFFFFFF00000000 still leaves room for 4 billion more insert operations, which may never be exceeded in your environment for decades to come. If your values are "reasonable", then, perhaps the best solution is to simply ignore the problem and deal with the one-off Status 146 issues that do arise. However, if your system has experienced repeated Status 146 issues, then it may trying to tell you something – you should do what it takes to eliminate synthetic time from your environment.

Searching for Files Using Synthetic Time

To address this problem in its entirety, you must eliminate all use of synthetic time within the environment (and keep it on real time from then onwards). The only way to do this is to first understand where the synthetic time is being generated from (which you now know after reading the first part of this document), and then to search every file in your environment to find files so affected.

Goldstar Software has released a free tool, called CheckSysData, which can handle this onerous task for you. You can find this tool and its documentation here:

http://www.goldstarsoftware.com/tools.asp

The easiest way to run the tool is to navigate to the root of your data volume and then to run the tool with the following command:

```
CheckSysData *.* /s
```

This will look at every file in the current directory and all subdirectories and attempt to locate any file where the highest System Data value is greater than 0x0900000000000000 (which is some time in the 2100's.) You will then have a list of files with synthetic time used for actual System Data records, and can decide if you are able to rebuild these files with new System Data values.

The above command is good, but it doesn't identify **every** file with problems. Were you paying attention enough to already see the shortcoming here? If a file cannot be opened or has no data records, then there will be no System Data values to examine. However, the LastInsertTimestamp may have already been set to a synthetic time, which can then impact other files! To address this, there is a more complete version of the scanner inside CheckSysData:

```
CheckSysData *.* /si
```

This command will not only look for the System Data values, but it will ALSO check the LastInsertTimestamp value. Note that Actian does not expose this field through normal means, so it has to be extracted by accessing the file directly through a read-only OS file handle. With Zen v15 though, this direct file access is blocked when the File Close Delay feature is enabled, which is why this check is not automatically enabled. For this to work as expected, please disable this feature (by setting File Close Delay to 0 in the ZenCC) and restart the engine to ensure that all files are closed.

In addition to scanning your data volume, you may ALSO wish to scan your boot/OS volume. At a bare minimum, though, you should scan the C:\ProgramData\Actian folder (or C:\ProgramData\Pervasive Software for older versions). We have seen cases where the synthetic time has migrated into the DBNAMES.CFG file, as well as DDF files inside the various system directories, and as you now know, attempting an insert into ANY table with a synthetic time will result in synthetic time being used again by the engine until the next restart. As such, it is critical to locate and address this issue in EVERY database file in your environment.

Eliminating Synthetic Time in the Environment

Once you know which files are using synthetic time, you can concentrate on the task of cleaning them up so that only real time is used internally. Luckily, this is the easy part!

When You Do Not Rely on System Data

If you do not rely on any System Data values for your environment (i.e. you are not using DataExchange or other replication solutions that are dependent on static values for each record), then the process to eliminate synthetic time is actually pretty easy:

1. Restart the database engine to ensure that you are working with real times.
2. Rebuild the affected data files using the -sK option to generate all new System Data values. While you *can* use the GUI Rebuild Utility to do this, you may find it far easier to use the command line rebuild utility (rbldcli) and wildcards to do entire folders at one time.

Note that the database files cannot be used while they are being rebuilt, so you will need to find a suitable system downtime window to complete this process. This fact also presents a problem with the special system files DBNAMES.CFG and the DEFAULTDB database DDF files, as these files are opened as part of opening any other file, so rebuilding will fail every time on these special files. To get around this inherent limitation, rename or copy these files to a new location, rebuild them there, and then copy them back.

Since you are rebuilding the files anyway, if you ever wanted to change file formats or move to a larger page size, now is the perfect time to consider that, too.

When You DO Rely on System Data

If you do rely on System Data values for your environment for things like replication, then things get infinitely more difficult. You cannot simply start rebuilding files willy-nilly, because the new System Data values won't exist in the related data files.

If you are running DataExchange, then you will need to re-deploy these files manually, which will likely require rebuilding the primary file, blanking the PDC file on the source server, blanking the PDC file and target file on the target server, and then running DXSyncTables. Alternatively, if MANY files are impacted, you may find it easier to tear down the DX environment and rebuild it from scratch, especially if many files are impacted. Note that this can create an extensive downtime window, so may also be important to reflect on the current synthetic time being used and determine if this will cause a problem any time soon.

Ensuring Synthetic Time Doesn't Come Back

Once you've spent the time to eliminate synthetic time from your environment, you need to be vigilant about making sure that it doesn't come back. This could mean scheduling periodic scans (annually?) to look for synthetic times used inside data files with CheckSysData.

If you periodically receive database files from other companies, then you may also want to be vigilant about scanning these files before accessing them on your system. If you are a developer and a customer sends you files with synthetic time, then your engine could flip to that time from that point forward, and you could then send files with synthetic time to other customers propagating the issue from site to site.

Alternatively, instead of a full system scan every so often, you can set up a process that periodically creates a new file (with System Data), inserts a new record into it, and then reads back the System Data value to ensure that it is within a normal time range. If you find a synthetic time here, then you know that your engine is impacted, and you can quickly schedule the downtime window needed to restart the engine and rebuild files once again. Of course, running a full scan at that point might be prudent, too, as even ONE file with a synthetic time can make it come back.

I'm so Confused!

As with most database issues, Goldstar Software is able to help you with anything you might need, so you can contact our support team for assistance with the analysis and clean-up.