Support for EPICS Time Stamps in asyn

Mark Rivers

September 9, 2013

Overview

There have been several requests to support EPICS time stamps in asyn port drivers and standard asyn device support. This will be used by setting the TSE field in the EPICS records to -2. The record support then directly uses the TIME field in the record as the timestamp. Device support must have properly set the TIME field in the record.

Setting the timestamp when the driver processes, rather than later on when the record processes can be desirable because the timestamp then reflects more accurately the actual time that the I/O operation was performed. It is also desirable to allow for a user-defined function to provide the timestamp, rather than being restricted to simply calling epicsTimeGetCurrent(). For example, the driver may be associated with a particular EPICS event, and the user-defined function would then call epicsTimeGetEvent() with that event ID. This can return a site-specific time format. For example at LCLS epicsTimeGetEvent() returns a timestamp where the low-order bits encode the pulse ID. They want drivers to read that timestamp as soon as possible after the I/O is complete.

The process of adding support for timestamps in asyn was begun by Eric Norum in asyn R4-20. This added a new "timestamp" field to the pasynUser structure. Standard asyn device support was changed to copy the pasynUser->timestamp field to precord->time. However, this was only supported for callbacks to device support, i.e. records with SCAN=I/O Intr, and not for records that called the read() function in the driver, i.e. records with SCAN!= I/O Intr. There was also no code in pasynManager or asynPortDriver to assist drivers in adding timestamps to the pasynUser structure on read operations or callbacks, or to support user-defined timestamp sources that could work with any driver. The changes described in this document address these issues.

asynManager changes

In order to support these requirements the following functions have been added to asynManager.

```
typedef void (*timeStampCallback)(void *userPvt, epicsTimeStamp *pTimeStamp);

asynStatus (*registerTimeStampSource)(asynUser *pasynUser, void *userPvt,
timeStampCallback callback);

asynStatus (*unregisterTimeStampSource)(asynUser *pasynUser);

asynStatus (*updateTimeStamp)(asynUser *pasynUser);

asynStatus (*getTimeStamp)(asynUser *pasynUser, epicsTimeStamp *pTimeStamp);

asynStatus (*setTimeStamp)(asynUser *pasynUser, const epicsTimeStamp
*pTimeStamp);
```

asynManager now has an epicsTimeStamp value that is stored for each asyn port. When the port is created there is a default function in asynManager to update this timestamp. This default function simply calls epicsTimeGetCurrent().

pasynManager->registerTimeStampSource() registers a user-defined function that will return an epicsTimeStamp each time that pasynManager->updateTimeStamp() is called.

pasynManager->unregisterTimeStampSource() unregisters any user-defined function and reverts to the default timestamp source in asynManager.

pasynManager->getTimeStamp() returns the internal time stamp returned by the last call to pasynManager->updateTimeStamp().

pasynManager->setTimeStamp() sets the internal timestamp to the value passed in this function.

The functions above provide the required infrastructure for asynManager to support driver timestamps, including user-defined timestamp sources. However, asynManager does not provide a means to set the pasynUser->timestamp field in read or callback operations. This is because asynManager is not involved in these operations; read calls are done directly from device support to the asyn port driver, and callback operations are done directly from the asyn port driver to device support.

asynPortDriver changes

The asynPortDriver C++ classes now provide very thin wrappers around several of the asynManager functions listed above. The following functions have been added to asynPortDriver:

```
virtual asynStatus updateTimeStamp();

virtual asynStatus getTimeStamp(epicsTimeStamp *pTimeStamp);

virtual asynStatus setTimeStamp(const epicsTimeStamp *pTimeStamp);
```

The functions in asynPortDriver that do callbacks to device support (triggered by callParamCallbacks(), doCallbacksXXXArray(), and doCallbacksGenericPointer()) now get the timestamp using asynPortDriver::getTimeStamp() and set the pasynUser->timestamp field in the callback.

In addition, all of the base class read() functions (e.g. readInt32(), readFloat64(), readUnt32Digital, readOctet) now also get the timestamp using asynPortDriver::getTimeStamp() and set the pasynUser->timestamp field.

Note that the asynPortDriver does not implement the readXXXArray() or readGenericPointer() functions, these are only implemented in derived classes. All derived classes should set pasynUser->timestamp and pasynUser->status in their readXXXArray() and readGenericPointer() methods.

Testing

The timestamp support must be implemented separately for each asyn interface (asynInt32, asynFloat64, asynInt8Array, etc.) in asynPortDriver. It must also be implemented separately for dozens of device support types (asynInt32 for ai, longin, bi, etc, asynInt32Array for waveform record,etc.). There are also code differences depending on whether the record is I/O Intr scanned or not. For these reasons it is important to have a comprehensive test program to verify that the timestamp support in asynManager, asynPortDriver, and standard asyn device support are all working properly.

There was already a test program in asyn to verify that the record STAT and SEVR fields were being set properly for most records types with standard asynDevice support, and for most asyn interfaces, with both periodically scanning and I/O Intr scanning. This is an asyn driver that derives from asynPortDriver and is located in asyn/testErrorsApp/src/testErrors.cpp. This driver and its example IOC (asyn/iocBoot/iocTestErrors) have been modified to also test the new timestamp support. The driver now calls asynPortDriver::updateTimeStamp() each time new values are computed, and it sets pasynUser->timestamp in the readXXXArray() methods.

testErrorsApp/src/testUserTimeStampSource is a new file that contains an example of a user-defined timestamp source. The timestamp source in that file simply calls epicsTimeGetCurrent(&timeStamp) and then sets timeStamp.nsec=0, so the time stamp is always an integer number of seconds. This makes it easy to tell that the user-defined timestamp source is being used. That file also defines two new example iocsh commands:

registerMyTimeStampSource(port)

and

unregisterMyTimeStampSource(port)

There are 3 settings in the IOC startup script that can be set for testing.

1) TSE in every record:

   0 (use the normal record timestamp mechanism)

   -2 (the record timestamp is set by device support)

2) SCAN in every record
   "2 second". Periodically scan the record

   "I/O Intr" Use callbacks from driver to trigger record processing

3) Time stamp source
   Default timestamp source in asynPortDriver

   User-defined timestamp source (timeStamp.nsec=0)

There are thus 8 tests that need to be done to verify that all combinations of the above parameters work correctly.

The record timestamps and record values were monitored with the “camonitor” program from epics base. asyn/iocBoot/iocTestErrors/timeStampMonitor.sh contains the camonitor commands to monitor all of the PVs in the test database.

Result 1. TSE=0, SCAN=2 second, user-defined timestamp source=No.

```
testErrors:AiInt32              2013-09-11 12:44:31.378018 1 12
testErrors:LonginInt32          2013-09-11 12:44:31.378024 1 12
testErrors:BiInt32              2013-09-11 12:44:31.378020 1 One
testErrors:MbbiInt32            2013-09-11 12:44:31.378028 1 Twelve STATE
MAJOR
testErrors:BiInt32              2013-09-11 12:44:31.378020 1 One
testErrors:LonginUInt32D        2013-09-11 12:44:31.378026 1 0
testErrors:BiUInt32D            2013-09-11 12:44:31.378022 1 Zero
testErrors:MbbiUInt32D          2013-09-11 12:44:31.378030 1 Zero
testErrors:MbbiDUInt32D         2013-09-11 12:44:31.378032 1 0
testErrors:AiFloat64            2013-09-11 12:44:31.378013 1 1.2
testErrors:SiOctet              2013-09-11 12:44:31.378036 1 1.2
testErrors:WfInOctet            2013-09-11 12:44:31.378044 1 49
testErrors:WfInt8               2013-09-11 12:44:31.378050 1 12
testErrors:WfInt16              2013-09-11 12:44:31.378046 1 12
testErrors:WfInt32              2013-09-11 12:44:31.378048 1 12
testErrors:WfFloat32            2013-09-11 12:44:31.378041 1 1.2
testErrors:WfFloat64            2013-09-11 12:44:31.378043 1 1.2
testErrors:AiFloat64            2013-09-11 12:44:33.378144 1 1.6
testErrors:AiInt32              2013-09-11 12:44:33.378155 1 0
testErrors:BiInt32              2013-09-11 12:44:33.378158 1 Zero
testErrors:BiInt32              2013-09-11 12:44:33.378158 1 Zero
testErrors:LonginInt32          2013-09-11 12:44:33.378164 1 0
testErrors:MbbiInt32            2013-09-11 12:44:33.378169 1 Zero
testErrors:SiOctet              2013-09-11 12:44:33.378178 1 1.6
testErrors:WfFloat32            2013-09-11 12:44:33.378183 1 1.6
testErrors:WfFloat64            2013-09-11 12:44:33.378186 1 1.6
testErrors:WfInOctet            2013-09-11 12:44:33.378188 1 49
testErrors:WfInt16              2013-09-11 12:44:33.378191 1 0
testErrors:WfInt32              2013-09-11 12:44:33.378193 1 0
testErrors:WfInt8               2013-09-11 12:44:33.378195 1 0
testErrors:AiFloat64            2013-09-11 12:44:35.378241 1 2
```

This test shows the expected normal behavior. Each record is processing every 2 seconds. The timestamps for each record in the group are slightly different because the records are processing sequentially in the 2 second scan task, and the timestamp is generated when the record processes.

Result 2. TSE=0, SCAN=2 second, user-defined timestamp source=Yes.

```
testErrors:AiInt32              2013-09-11 12:52:02.623938 1 8
testErrors:LonginInt32          2013-09-11 12:52:02.623944 1 8
testErrors:BiInt32              2013-09-11 12:52:02.623940 1 One
testErrors:MbbiInt32            2013-09-11 12:52:02.623948 1 Eight STATE MINOR
testErrors:BiInt32              2013-09-11 12:52:02.623940 1 One
testErrors:LonginUInt32D        2013-09-11 12:52:02.623946 1 0
testErrors:BiUInt32D            2013-09-11 12:52:02.623942 1 Zero
testErrors:MbbiUInt32D          2013-09-11 12:52:02.623950 1 Zero
testErrors:MbbiDUInt32D         2013-09-11 12:52:02.623952 1 0
testErrors:AiFloat64            2013-09-11 12:52:02.623934 1 0.8
testErrors:SiOctet              2013-09-11 12:52:02.623956 1 0.8
testErrors:WfInOctet            2013-09-11 12:52:02.623964 1 48
testErrors:WfInt8               2013-09-11 12:52:02.623970 1 8
testErrors:WfInt16              2013-09-11 12:52:02.623966 1 8
testErrors:WfInt32              2013-09-11 12:52:02.623968 1 8
testErrors:WfFloat32            2013-09-11 12:52:02.623960 1 0.8
testErrors:WfFloat64            2013-09-11 12:52:02.623962 1 0.8
testErrors:AiFloat64            2013-09-11 12:52:04.624023 1 1.2
testErrors:AiInt32              2013-09-11 12:52:04.624035 1 12
testErrors:LonginInt32          2013-09-11 12:52:04.624044 1 12
testErrors:MbbiInt32            2013-09-11 12:52:04.624048 1 Twelve STATE
MAJOR
testErrors:SiOctet              2013-09-11 12:52:04.624058 1 1.2
testErrors:WfFloat32            2013-09-11 12:52:04.624066 1 1.2
testErrors:WfFloat64            2013-09-11 12:52:04.624071 1 1.2
testErrors:WfInOctet            2013-09-11 12:52:04.624073 1 49
testErrors:WfInt16              2013-09-11 12:52:04.624075 1 12
testErrors:WfInt32              2013-09-11 12:52:04.624079 1 12
testErrors:WfInt8               2013-09-11 12:52:04.624083 1 12
testErrors:AiFloat64            2013-09-11 12:52:06.624110 1 1.6
testErrors:AiInt32              2013-09-11 12:52:06.624122 1 0
```

As expected this test shows the same results as test 1, because setting a user-defined timestamp source should have no effect when TSE=0.

Result 3. TSE=0, SCAN=I/O Intr, user-defined timestamp source=No.

```
testErrors:AiInt32              2013-09-11 12:57:55.453886 1 6
testErrors:LonginInt32          2013-09-11 12:57:55.453891 1 6
testErrors:BiInt32              2013-09-11 12:57:55.453890 1 One
testErrors:MbbiInt32            2013-09-11 12:57:55.453892 1 Six STATE MINOR
testErrors:BiInt32              2013-09-11 12:57:55.453890 1 One
testErrors:LonginUInt32D        2013-09-11 12:57:53.053579 1 0
testErrors:BiUInt32D            2013-09-11 12:57:53.053578 1 Zero
testErrors:MbbiUInt32D          2013-09-11 12:57:53.053581 1 Zero
testErrors:MbbiDUInt32D         2013-09-11 12:57:53.053584 1 0
testErrors:AiFloat64            2013-09-11 12:57:55.453894 1 0.6
testErrors:SiOctet              2013-09-11 12:57:55.453895 1 0.6
testErrors:WfInOctet            2013-09-11 12:57:55.453897 1 48
testErrors:WfInt8               2013-09-11 12:57:55.453899 1 6
testErrors:WfInt16              2013-09-11 12:57:55.453900 1 6
testErrors:WfInt32              2013-09-11 12:57:55.453901 1 6
testErrors:WfFloat32            2013-09-11 12:57:55.453902 1 0.6
testErrors:WfFloat64            2013-09-11 12:57:55.453903 1 0.6
testErrors:AiInt32              2013-09-11 12:57:55.954073 1 7
testErrors:LonginInt32          2013-09-11 12:57:55.954086 1 7
testErrors:MbbiInt32            2013-09-11 12:57:55.954088 1 Seven STATE MINOR
testErrors:AiFloat64            2013-09-11 12:57:55.954090 1 0.7
testErrors:SiOctet              2013-09-11 12:57:55.954092 1 0.7
testErrors:WfInOctet            2013-09-11 12:57:55.954095 1 48
testErrors:WfInt8               2013-09-11 12:57:55.954098 1 7
testErrors:WfInt16              2013-09-11 12:57:55.954099 1 7
testErrors:WfInt32              2013-09-11 12:57:55.954101 1 7
testErrors:WfFloat32            2013-09-11 12:57:55.954102 1 0.7
testErrors:WfFloat64            2013-09-11 12:57:55.954103 1 0.7
testErrors:AiInt32              2013-09-11 12:57:56.454213 1 8
```

This test shows the expected normal behavior. Each record is processing every 0.5 seconds because that is the delay in the driver thread doing the callbacks. The timestamps for each record in the group are slightly different because the records are processing sequentially and the timestamp is generated when the record processes.

Result 4. TSE=0, SCAN=I/O Intr, user-defined timestamp source=Yes.

```
testErrors:AiInt32              2013-09-11 13:03:15.191708 1 0
testErrors:BiInt32              2013-09-11 13:03:15.191718 1 Zero
testErrors:BiInt32              2013-09-11 13:03:15.191718 1 Zero
testErrors:LonginInt32          2013-09-11 13:03:15.191721 1 0
testErrors:MbbiInt32            2013-09-11 13:03:15.191723 1 Zero
testErrors:AiFloat64            2013-09-11 13:03:15.191725 1 1.6
testErrors:SiOctet              2013-09-11 13:03:15.191727 1 1.6
testErrors:WfInOctet            2013-09-11 13:03:15.191729 1 49
testErrors:WfInt8               2013-09-11 13:03:15.191732 1 0
testErrors:WfInt16              2013-09-11 13:03:15.191733 1 0
testErrors:WfInt32              2013-09-11 13:03:15.191734 1 0
testErrors:WfFloat32            2013-09-11 13:03:15.191736 1 1.6
testErrors:WfFloat64            2013-09-11 13:03:15.191737 1 1.6
testErrors:AiInt32              2013-09-11 13:03:15.691816 1 1
testErrors:BiInt32              2013-09-11 13:03:15.691825 1 One
testErrors:BiInt32              2013-09-11 13:03:15.691825 1 One
testErrors:LonginInt32          2013-09-11 13:03:15.691828 1 1
testErrors:MbbiInt32            2013-09-11 13:03:15.691830 1 One
testErrors:AiFloat64            2013-09-11 13:03:15.691832 1 1.7
testErrors:SiOctet              2013-09-11 13:03:15.691833 1 1.7
testErrors:WfInOctet            2013-09-11 13:03:15.691836 1 49
testErrors:WfInt8               2013-09-11 13:03:15.691839 1 1
testErrors:WfInt16              2013-09-11 13:03:15.691840 1 1
testErrors:WfInt32              2013-09-11 13:03:15.691841 1 1
testErrors:WfFloat32            2013-09-11 13:03:15.691842 1 1.7
testErrors:WfFloat64            2013-09-11 13:03:15.691843 1 1.7
testErrors:AiInt32              2013-09-11 13:03:16.192034 1 2
```

As expected this test shows the same results as test 3, because setting a user-defined timestamp source should have no effect when TSE=0.

Result 5. TSE=-2, SCAN=2 second, user-defined timestamp source=No.

```
testErrors:AiInt32              2013-09-11 13:10:02.410217 1 12
testErrors:LonginInt32          2013-09-11 13:10:02.410217 1 12
testErrors:BiInt32              2013-09-11 13:10:02.410217 1 One
testErrors:MbbiInt32            2013-09-11 13:10:02.410217 1 Twelve STATE
MAJOR
testErrors:BiInt32              2013-09-11 13:10:02.410217 1 One
testErrors:LonginUInt32D        2013-09-11 13:10:02.410217 1 0
testErrors:BiUInt32D            2013-09-11 13:10:02.410217 1 Zero
testErrors:MbbiUInt32D          2013-09-11 13:10:02.410217 1 Zero
testErrors:MbbiDUInt32D         2013-09-11 13:10:02.410217 1 0
testErrors:AiFloat64            2013-09-11 13:10:02.410217 1 1.2
testErrors:SiOctet              2013-09-11 13:10:02.410217 1 1.2
testErrors:WfInOctet            2013-09-11 13:10:02.410217 1 49
testErrors:WfInt8               2013-09-11 13:10:02.410217 1 12
testErrors:WfInt16              2013-09-11 13:10:02.410217 1 12
testErrors:WfInt32              2013-09-11 13:10:02.410217 1 12
testErrors:WfFloat32            2013-09-11 13:10:02.410217 1 1.2
testErrors:WfFloat64            2013-09-11 13:10:02.410217 1 1.2
testErrors:AiFloat64            2013-09-11 13:10:04.410734 1 1.6
testErrors:AiInt32              2013-09-11 13:10:04.410734 1 0
testErrors:BiInt32              2013-09-11 13:10:04.410734 1 Zero
testErrors:BiInt32              2013-09-11 13:10:04.410734 1 Zero
testErrors:LonginInt32          2013-09-11 13:10:04.410734 1 0
testErrors:MbbiInt32            2013-09-11 13:10:04.410734 1 Zero
testErrors:SiOctet              2013-09-11 13:10:04.410734 1 1.6
testErrors:WfFloat32            2013-09-11 13:10:04.410734 1 1.6
testErrors:WfFloat64            2013-09-11 13:10:04.410734 1 1.6
testErrors:WfInOctet            2013-09-11 13:10:04.410734 1 49
testErrors:WfInt16              2013-09-11 13:10:04.410734 1 0
testErrors:WfInt32              2013-09-11 13:10:04.410734 1 0
testErrors:WfInt8               2013-09-11 13:10:04.410734 1 0
testErrors:AiFloat64            2013-09-11 13:10:06.411245 1 2
testErrors:AiInt32              2013-09-11 13:10:06.411245 1 4
```

This test shows the expected result. The timestamps within a single 2-second group are identical, which is different from test 1. This is because the timestamps are coming from the driver, reflecting the most recent call by the driver to updateTimeStamp(). The timestamps have non-zero nsec field because the default timestamp source in pasynManager is being used.

Result 6. TSE=-2, SCAN=2 second, user-defined timestamp source=Yes.

```
testErrors:AiInt32              2013-09-11 13:15:31.000000 1 8
testErrors:LonginInt32          2013-09-11 13:15:31.000000 1 8
testErrors:BiInt32              2013-09-11 13:15:31.000000 1 One
testErrors:MbbiInt32            2013-09-11 13:15:31.000000 1 Eight STATE MINOR
testErrors:BiInt32              2013-09-11 13:15:31.000000 1 One
testErrors:LonginUInt32D        2013-09-11 13:15:31.000000 1 0
testErrors:BiUInt32D            2013-09-11 13:15:31.000000 1 Zero
testErrors:MbbiUInt32D          2013-09-11 13:15:31.000000 1 Zero
testErrors:MbbiDUInt32D         2013-09-11 13:15:31.000000 1 0
testErrors:AiFloat64            2013-09-11 13:15:31.000000 1 0.8
testErrors:SiOctet              2013-09-11 13:15:31.000000 1 0.8
testErrors:WfInOctet            2013-09-11 13:15:31.000000 1 48
testErrors:WfInt8               2013-09-11 13:15:31.000000 1 8
testErrors:WfInt16              2013-09-11 13:15:31.000000 1 8
testErrors:WfInt32              2013-09-11 13:15:31.000000 1 8
testErrors:WfFloat32            2013-09-11 13:15:31.000000 1 0.8
testErrors:WfFloat64            2013-09-11 13:15:31.000000 1 0.8
testErrors:AiFloat64            2013-09-11 13:15:33.000000 1 1.2
testErrors:AiInt32              2013-09-11 13:15:33.000000 1 12
testErrors:LonginInt32          2013-09-11 13:15:33.000000 1 12
testErrors:MbbiInt32            2013-09-11 13:15:33.000000 1 Twelve STATE
MAJOR
testErrors:SiOctet              2013-09-11 13:15:33.000000 1 1.2
testErrors:WfFloat32            2013-09-11 13:15:33.000000 1 1.2
testErrors:WfFloat64            2013-09-11 13:15:33.000000 1 1.2
testErrors:WfInOctet            2013-09-11 13:15:33.000000 1 49
testErrors:WfInt16              2013-09-11 13:15:33.000000 1 12
testErrors:WfInt32              2013-09-11 13:15:33.000000 1 12
testErrors:WfInt8               2013-09-11 13:15:33.000000 1 12
testErrors:AiFloat64            2013-09-11 13:15:35.000000 1 1.6
testErrors:AiInt32              2013-09-11 13:15:35.000000 1 0
```

This test shows the expected result. The timestamps within a single 2-second group are identical, which is different from test 2. This is because the timestamps are coming from the driver, reflecting the most recent call by the driver to updateTimeStamp(). The timestamps have no fractional seconds because the user-defined timestamp source is being used, which sets timeStamp.nsec=0.

Result 7. TSE=-2, SCAN=I/O Intr, user-defined timestamp source=No.

```
testErrors:AiInt32              2013-09-11 13:18:40.790244 1 6
testErrors:LonginInt32          2013-09-11 13:18:40.790244 1 6
testErrors:MbbiInt32            2013-09-11 13:18:40.790244 1 Six STATE MINOR
testErrors:AiFloat64            2013-09-11 13:18:40.790244 1 0.6
testErrors:SiOctet              2013-09-11 13:18:40.790244 1 0.6
testErrors:WfInOctet            2013-09-11 13:18:40.790244 1 48
testErrors:WfInt8               2013-09-11 13:18:40.790244 1 6
testErrors:WfInt16              2013-09-11 13:18:40.790244 1 6
testErrors:WfInt32              2013-09-11 13:18:40.790244 1 6
testErrors:WfFloat32            2013-09-11 13:18:40.790244 1 0.6
testErrors:WfFloat64            2013-09-11 13:18:40.790244 1 0.6
testErrors:AiInt32              2013-09-11 13:18:41.290368 1 7
testErrors:LonginInt32          2013-09-11 13:18:41.290368 1 7
testErrors:MbbiInt32            2013-09-11 13:18:41.290368 1 Seven STATE MINOR
testErrors:AiFloat64            2013-09-11 13:18:41.290368 1 0.7
testErrors:SiOctet              2013-09-11 13:18:41.290368 1 0.7
testErrors:WfInOctet            2013-09-11 13:18:41.290368 1 48
testErrors:WfInt8               2013-09-11 13:18:41.290368 1 7
testErrors:WfInt16              2013-09-11 13:18:41.290368 1 7
testErrors:WfInt32              2013-09-11 13:18:41.290368 1 7
testErrors:WfFloat32            2013-09-11 13:18:41.290368 1 0.7
testErrors:WfFloat64            2013-09-11 13:18:41.290368 1 0.7
testErrors:AiInt32              2013-09-11 13:18:41.790539 1 8
```

This test shows the expected result. The timestamps within a single 0.5 second group are identical, which is different from test 3. This is because the timestamps are coming from the driver, reflecting the most recent call by the driver to updateTimeStamp(). The timestamps have non-zero nsec field because the default timestamp source in pasynManager is being used.

Result 8. TSE=-2, SCAN=I/O Intr, user-defined timestamp source=Yes.

```
testErrors:AiInt32              2013-09-11 13:22:55.000000 1 0
testErrors:BiInt32              2013-09-11 13:22:55.000000 1 Zero
testErrors:BiInt32              2013-09-11 13:22:55.000000 1 Zero
testErrors:LonginInt32          2013-09-11 13:22:55.000000 1 0
testErrors:MbbiInt32            2013-09-11 13:22:55.000000 1 Zero
testErrors:AiFloat64            2013-09-11 13:22:55.000000 1 1.6
testErrors:SiOctet              2013-09-11 13:22:55.000000 1 1.6
testErrors:WfInOctet            2013-09-11 13:22:55.000000 1 49
testErrors:WfInt8               2013-09-11 13:22:55.000000 1 0
testErrors:WfInt16              2013-09-11 13:22:55.000000 1 0
testErrors:WfInt32              2013-09-11 13:22:55.000000 1 0
testErrors:WfFloat32            2013-09-11 13:22:55.000000 1 1.6
testErrors:WfFloat64            2013-09-11 13:22:55.000000 1 1.6
testErrors:AiInt32              2013-09-11 13:22:55.000000 1 1
testErrors:BiInt32              2013-09-11 13:22:55.000000 1 One
testErrors:BiInt32              2013-09-11 13:22:55.000000 1 One
testErrors:LonginInt32          2013-09-11 13:22:55.000000 1 1
testErrors:MbbiInt32            2013-09-11 13:22:55.000000 1 One
testErrors:AiFloat64            2013-09-11 13:22:55.000000 1 1.7
testErrors:SiOctet              2013-09-11 13:22:55.000000 1 1.7
testErrors:WfInOctet            2013-09-11 13:22:55.000000 1 49
testErrors:WfInt8               2013-09-11 13:22:55.000000 1 1
testErrors:WfInt16              2013-09-11 13:22:55.000000 1 1
testErrors:WfInt32              2013-09-11 13:22:55.000000 1 1
testErrors:WfFloat32            2013-09-11 13:22:55.000000 1 1.7
testErrors:WfFloat64            2013-09-11 13:22:55.000000 1 1.7
testErrors:AiInt32              2013-09-11 13:22:56.000000 1 2
t
```

This test shows the expected result. The timestamps within a single 0.5 second group are identical, which is different from test 4. This is because the timestamps are coming from the driver, reflecting the most recent call by the driver to updateTimeStamp(). The timestamps have no fractional seconds because the user-defined timestamp source is being used, which sets timeStamp.nsec=0. Because the records are processing at 0.5 seconds two sets of record processing have the same timestamp, since it has 1 second resolution.

Result 9. TSE=-2, SCAN=I/O Intr, user-defined timestamp source=Yes.

```
testErrors:WfInt32              2013-09-11 13:33:31.000000 1 13
testErrors:WfFloat32            2013-09-11 13:33:31.000000 1 4.5
testErrors:WfFloat64            2013-09-11 13:33:31.000000 1 4.5
testErrors:AiInt32              2013-09-11 13:33:31.796358 1 14
testErrors:LonginInt32          2013-09-11 13:33:31.796358 1 14
testErrors:MbbiInt32            2013-09-11 13:33:31.796358 1 Fourteen STATE
INVALID
testErrors:AiFloat64            2013-09-11 13:33:31.796358 1 4.6
testErrors:SiOctet              2013-09-11 13:33:31.796358 1 4.6
testErrors:WfInOctet            2013-09-11 13:33:31.796358 1 52
testErrors:WfInt8               2013-09-11 13:33:31.796358 1 14
testErrors:WfInt16              2013-09-11 13:33:31.796358 1 14
testErrors:WfInt32              2013-09-11 13:33:31.796358 1 14
testErrors:WfFloat32            2013-09-11 13:33:31.796358 1 4.6
testErrors:WfFloat64            2013-09-11 13:33:31.796358 1 4.6
testErrors:AiInt32              2013-09-11 13:33:32.296489 1 15
testErrors:LonginInt32          2013-09-11 13:33:32.296489 1 15
testErrors:MbbiInt32            2013-09-11 13:33:32.296489 1 Fifteen STATE
INVALID
testErrors:AiFloat64            2013-09-11 13:33:32.296489 1 4.7
testErrors:SiOctet              2013-09-11 13:33:32.296489 1 4.7
testErrors:WfInOctet            2013-09-11 13:33:32.296489 1 52
testErrors:WfInt8               2013-09-11 13:33:32.296489 1 15
testErrors:WfInt16              2013-09-11 13:33:32.296489 1 15
testErrors:WfInt32              2013-09-11 13:33:32.296489 1 15
testErrors:WfFloat32            2013-09-11 13:33:32.296489 1 4.7
testErrors:WfFloat64            2013-09-11 13:33:32.296489 1 4.7
testErrors:AiInt32              2013-09-11 13:33:32.000000 1 0
testErrors:BiInt32              2013-09-11 13:33:32.000000 1 Zero
testErrors:BiInt32              2013-09-11 13:33:32.000000 1 Zero
```

The above output shows the result when the following commands are typed at the IOC shell.

```
unregisterMyTimeStampSource(PORT1)
registerMyTimeStampSource(PORT1)
unregisterMyTimeStampSource(PORT1)
```

The timestamps switch between being the user-defined timestamp (no fractional seconds) and the default timestamp source (with fractional seconds). This shows that timestamp source can be changed dynamically at run-time.

Future plans

One of the requirements of LCLS is that areaDetector should support user-defined timestamps. The plan is to implement this as follows:

The NDArray class currently has a timeStamp field. This field is a double however, not an epicsTimeStamp. The double field is useful because it is streamed to disk files (netCDF, Nexus, HDF5) and file readers can easily use it. However, it cannot store all of the information in an epicsTimeStamp, so a new field, epicsTimeStamp will be added to NDArray.

When areaDetector drivers collect a new image they will call updateTimeStamp(), and then getTimeStamp(&pArray->epicsTimeStamp) to put the time stamp in the NDArray. This can be a user-defined timestamp source and format if a user-defined timestamp source has been configured.

File writing plugins will be changed to also write the pArray->epicsTime field to disk.

All plugins will call setTimeStamp(&pArray->epicsTimeStamp). This will set the current timestamp of the plugin driver to the value when the image was collected. All records that process from the plugin will thus have that timestamp, not the current time when the record processed.

All of this should be backwards compatible because it only affects records with TSE=-2, and that was not previously supported in areaDetector.