Channel Access Servers

Kenneth Evans, Jr.
October 8, 2004

Part of the EPICS "Getting Started" Lecture Series

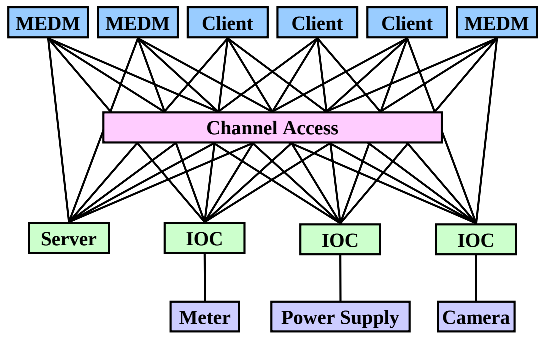
Argonne National Laboratory

A U.S. Department of Energy
Office of Science Laboratory
Operated by The University of Chicago

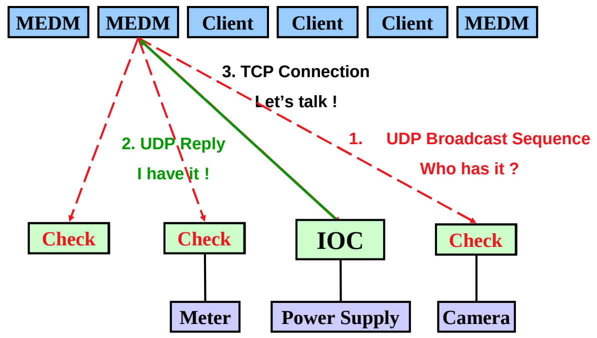
Outline

- Channel Access Concepts
- Types of Servers
- Server Examples
- Server Application Programming Interface [API]

EPICS Overview



Search and Connect Procedure



Search Request

- **A search request consists of a sequence of UDP packets**
  - Only goes to EPICS_CA_ADDR_LIST
  - Starts with a small interval (30 ms), that doubles each time
  - Until it gets larger than 5 s, then it stays at 5 s
  - Stops after 100 packets or when it gets a response
  - Never tries again until it sees a beacon anomaly or creates a new PV (Continues to search at a lower rate in 3.14.7)
  - Total time is about 8 minutes to do all 100

- **Servers have to do an Exist Test for each packet**
- **Usually connects on the first packet or the first few**
- **Non-existent PVs cause a lot of traffic**
  - Try to eliminate them

Beacons

- **A Beacon is a UDP broadcast packet sent by a Server**
- **When it is healthy, each Server broadcasts a UDP beacon at regular intervals (like a heartbeat)**
  - EPICS_CA_BEACON_PERIOD, 15 s by default

- **When it is coming up, each Server broadcasts a startup sequence of UDP beacons**
  - Starts with a small interval (25 ms, 75 ms for VxWorks)
  - Interval doubles each time
  - Until it gets larger than 15 s, then it stays at 15 s
    - *Takes about 10 beacons and 40 s to get to steady state*

- **Clients monitor the beacons**
  - Determine connection status, whether to reissue searches

Environment Variables

• CA Client	• CA Server
EPICS_CA_ADDR_LIST	EPICS_CAS_SERVER_PORT
EPICS_CA_AUTO_ADDR_LIST	EPICS_CAS_AUTO_BEACON_ADDR_LIST
EPICS_CA_CONN_TMO	EPICS_CAS_BEACON_ADDR_LIST
EPICS_CA_BEACON_PERIOD	EPICS_CAS_BEACON_PERIOD
EPICS_CA_REPEATER_PORT	EPICS_CAS_BEACON_PORT
EPICS_CA_SERVER_PORT	EPICS_CAS_INTF_ADDR_LIST
EPICS_CA_MAX_ARRAY_BYTES	EPICS_CAS_IGNORE_ADDR_LIST
EPICS_TS_MIN_WEST	

- **See the Channel Access Reference Manual for more information**

Channel Access Reference Manual

- **The place to go for more information**
- **Found in the EPICS web pages**
  - http://www.aps.anl.gov/epics/index.php
  - Look under Documents
  - Also under Base, then a specific version of Base
- **There is not as much information on servers as on clients**

Types of Servers

- **RSRV: Server for IOCs and Soft IOCs**
  - IOCs: Typically run in instrument crates
  - Soft IOCs: Typically run on workstations
  - Runs iocCore and uses the EPICS database
    - *Has records, fields, etc.*
    - *Described in the next set of lectures*
- **CAS: Channel Access Server or Portable server**
  - Runs on workstations
  - Plan is to have it replace RSRV so there is only one server
  - Has whatever process variables you want to provide
    - *Probably does not have records, fields, etc.*
    - *Could have a different naming convention*
      - xxx.yyy.zzz%VAL

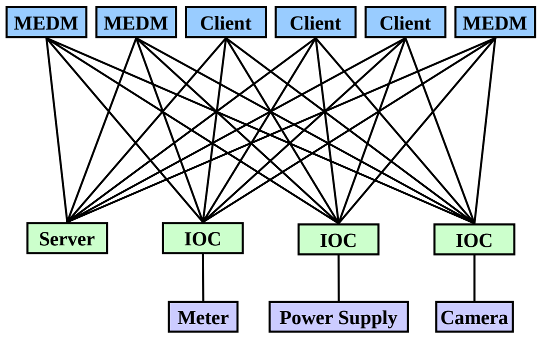
Soft IOCs

- **Soft IOCs may be a better choice than CAS**
- **Easier to implement**
  - Are essentially the same as IOCs
  - Except they run on workstations
- **Better documented**
- **Have the database, records, and fields of a real IOC**
- **And you don't have to write and maintain an application**

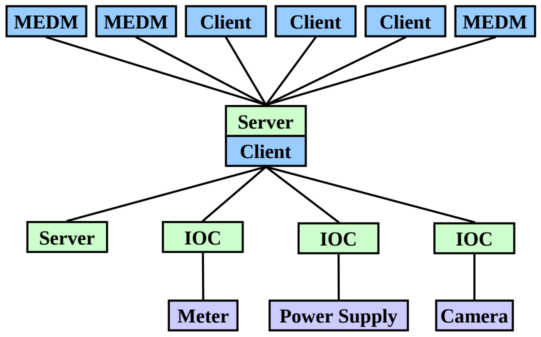
CAS Examples

- **Gateway**
  - Both a CAC and a CAS
  - Gets its process variables from other servers (CAC part)
  - Serves them to clients (CAS part)
- **CaSnooper**
  - Keeps track of Exist Tests
  - Prints reports about them
  - Used to track nonexistent process variables
  - Publishes some process variables
- **Excas**
  - Sample server
  - Comes with base

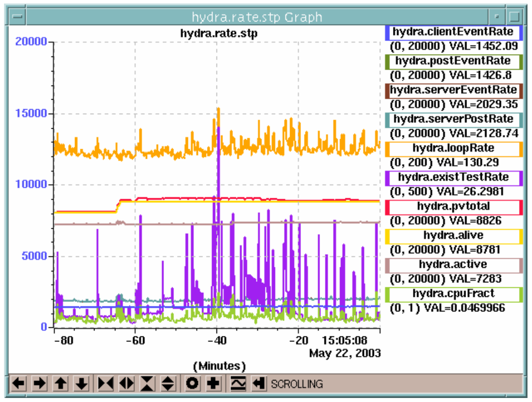
No Gateway



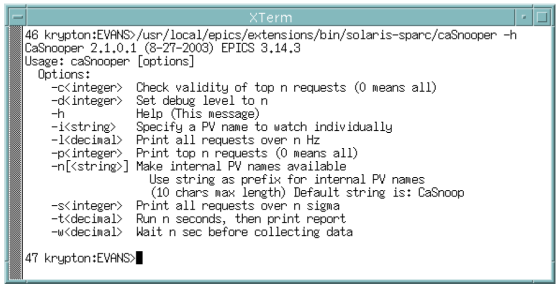
Gateway



Diagnostics via Internal PVs

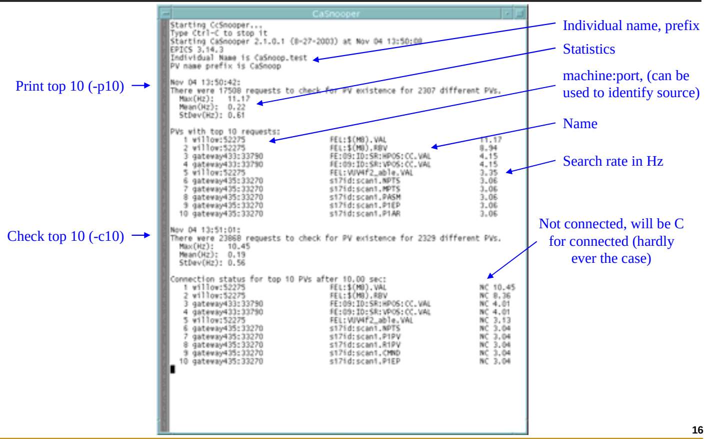


CaSnooper



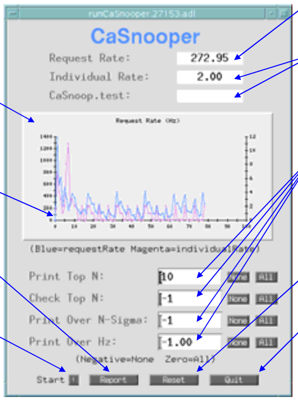
```
46 krypton:EVANS>/usr/local/epics/extensions/bin/solaris-sparc/caSnooper -h
CaSnooper 2.1.0.1 (8-27-2003) EPICS 3.14.3
Usage: caSnooper [options]
  Options:
    -c<integer>   Check validity of top n requests (0 means all)
    -d<integer>   Set debug level to n
    -h            Help (This message)
    -i<string>    Specify a PV name to watch individually
    -l<decimal>   Print all requests over n Hz
    -p<integer>   Print top n requests (0 means all)
    -n[<string>]  Make internal PV names available
                    Use string as prefix for internal PV names
                    (10 chars max length) Default string is: CaSnoop
    -s<integer>   Print all requests over n sigma
    -t<decimal>   Run n seconds, then print report
    -w<decimal>   Wait n sec before collecting data

47 krypton:EVANS>█
```

Sample CaSnooper Output


CaSnooper
Individual name, prefix
Statistics
machine:port, (can be used to identify source)
Name
Search rate in Hz
Print top 10 (-p10)
Check top 10 (-c10)
Not connected, will be C for connected (hardly ever the case)

```
Starting CcSnooper...
Type Ctrl-C to stop it
Starting CaSnooper 2.1.0.1 (8-27-2003) at Nov 04 13:50:08
EPICS 3.14.3
Individual Name is CaSnoop.test
PV name prefix is CaSnoop

Nov 04 13:50:42:
There were 17508 requests to check for PV existence for 2307 different PVs.
  Max(Hz):    11.17
  Mean(Hz):   0.22
  StDev(Hz):  0.61

PVs with top 10 requests:
   1 willow:52275           FEL:$(M8).VAL                 11.17
   2 willow:52275           FEL:$(M8).RBV                 8.94
   3 gateway433:33790       FE:09:ID:SR:HPOS:CC.VAL       4.15
   4 gateway433:33790       FE:09:ID:SR:VPOS:CC.VAL       4.15
   5 willow:52275           FEL:VUV4f2_able.VAL           3.35
   6 gateway435:33270       s17id:scan1.NPTS              3.06
   7 gateway435:33270       s17id:scan1.MPTS              3.06
   8 gateway435:33270       s17id:scan1.PASM              3.06
   9 gateway435:33270       s17id:scan1.P1EP              3.06
  10 gateway435:33270       s17id:scan1.P1AR              3.06

Nov 04 13:51:01:
There were 23868 requests to check for PV existence for 2329 different PVs.
  Max(Hz):    10.45
  Mean(Hz):   0.19
  StDev(Hz):  0.56

Connection status for top 10 PVs after 10.00 sec:
   1 willow:52275           FEL:$(M8).VAL                 NC 10.45
   2 willow:52275           FEL:$(M8).RBV                 NC 8.36
   3 gateway433:33790       FE:09:ID:SR:HPOS:CC.VAL       NC 4.01
   4 gateway433:33790       FE:09:ID:SR:VPOS:CC.VAL       NC 4.01
   5 willow:52275           FEL:VUV4f2_able.VAL           NC 3.13
   6 gateway435:33270       s17id:scan1.NPTS              NC 3.04
   7 gateway435:33270       s17id:scan1.P1PV              NC 3.04
   8 gateway435:33270       s17id:scan1.R1PV              NC 3.04
   9 gateway435:33270       s17id:scan1.CMND              NC 3.04
  10 gateway435:33270       s17id:scan1.P1EP              NC 3.04
█
```

Control CaSnooper via MEDM



runCaSnooper.27153.adl

CaSnooper

```
Request Rate:          272.95
Individual Rate:         2.00
CaSnoop.test:
```

```
Print Top N:          10      None  All
Check Top N:          -1      None  All
Print Over N-Sigma:   -1      None  All
Print Over Hz:        -1.00   None  All
      (Negative=None  Zero=All)
Start   Report   Reset   Quit
```

Request rate

Individual rate for CaSnoop.test, which doesn't exist

Cartesian plot of requestRate and individualRate

CaSnooper was started here (with EPICS_CA_REPEATER_PORT = default=5065)

Use these to set what will happen when you press Report. Case illustrated will print the top 10.

Execute selected reports in the CaSnooper stdout

Reset the counters in CaSnooper

Shell command to start CaSnooper, MEDM, StripTool, etc.

Stop CaSnooper

Excas

- **Stands for Example Channel Access Server**
- **Source files are in**
  - epics/base/templates/makeBaseApp/top/caServerApp
  - See the README file there
- **Should also be an executable in**
  - epics/base/bin/solaris-sparc/excas
- **Usage**

```
excas [-d<debug level> -t<execution time>
-p<PV name prefix> -c<numbered alias count>
-s<1=scan on (default), 0=scan off>
-ss<1=synchronous scan (default), 0=asynchronous scan>]
```

- **Example**

```
excas –pevans:
```

- **Makes a good starting point for your server**

Making a Project for Excas

- **Make a standalone project outside the build system**

```
cd example
perl epics/base/bin/solaris-sparc/makeBaseApp.pl
  -t caServerApp myserver
```

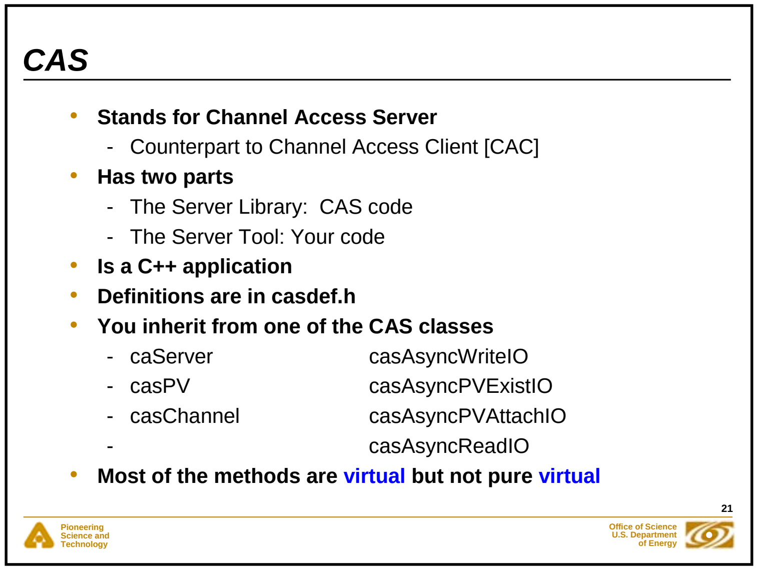
  - Do a make in example (It also makes example/myserverApp)
  - Executable is example/bin/solaris-sparc/casexample
- **Make an extension**

```
cd epics/extensions/src/excas
cp epics/base/templates/makeBaseApp/top/
  caServerApp/* .
```

  - In Makefile change "**`TOP=..`**" to "**`TOP=../..`**"
  - Do a make in excas
  - Excutable is in excas/O.solaris-sparc/casexample
- **Change `PROD_HOST` in Makefile to excas if you like**

Excas Process Variables

Scan Period	Name	HOPR	LOPR	Type	Async	Count
.1	"jane"	10.0	0.0	DBR_DOUBLE	No	1
2.0	"fred"	10.0	-10.0	DBR_DOUBLE	No	1
.1	"janet"	10.0	0.0	DBR_DOUBLE	Yes	1
2.0	"freddy"	10.0	-10.0	DBR_DOUBLE	Yes	1
2.0	"alan"	10.0	-10.0	DBR_DOUBLE	No	100
20.0	"albert"	10.0	-10.0	DBR_DOUBLE	No	1000
-1.0	"boot"	10.0	-10.0	DBR_ENUM	No	1
-1.0	"booty"	10.0	-10.0	DBR_ENUM	Yes	1
-1.0	"bill"	10.0	-10.0	DBR_DOUBLE	No	1
-1.0	"billy"	10.0	-10.0	DBR_DOUBLE	Yes	1
-1.0	"bloaty"	10.0	-10.0	DBR_DOUBLE	No	100000



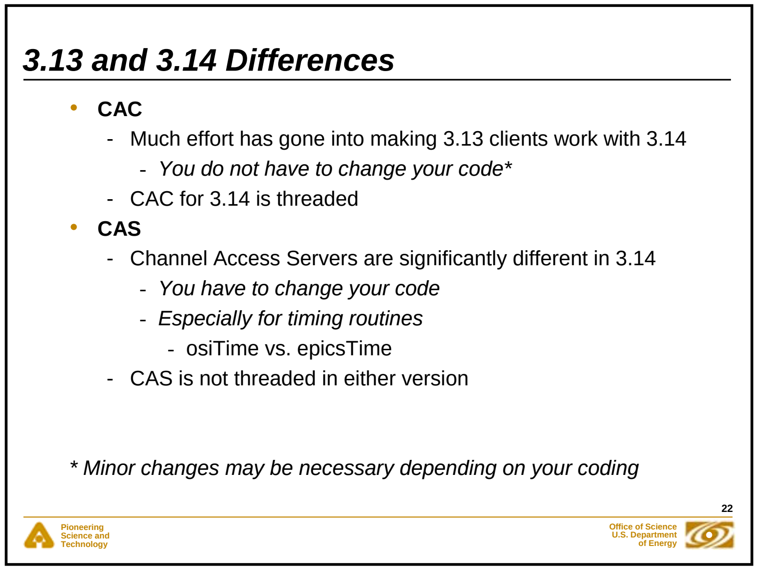
CAS

- **Stands for Channel Access Server**
  - Counterpart to Channel Access Client [CAC]
- **Has two parts**
  - The Server Library: CAS code
  - The Server Tool: Your code
- **Is a C++ application**
- **Definitions are in casdef.h**
- **You inherit from one of the CAS classes**

caServer	casAsyncWriteIO
casPV	casAsyncPVExistIO
casChannel	casAsyncPVAttachIO
	casAsyncReadIO

- **Most of the methods are virtual but not pure virtual**

21

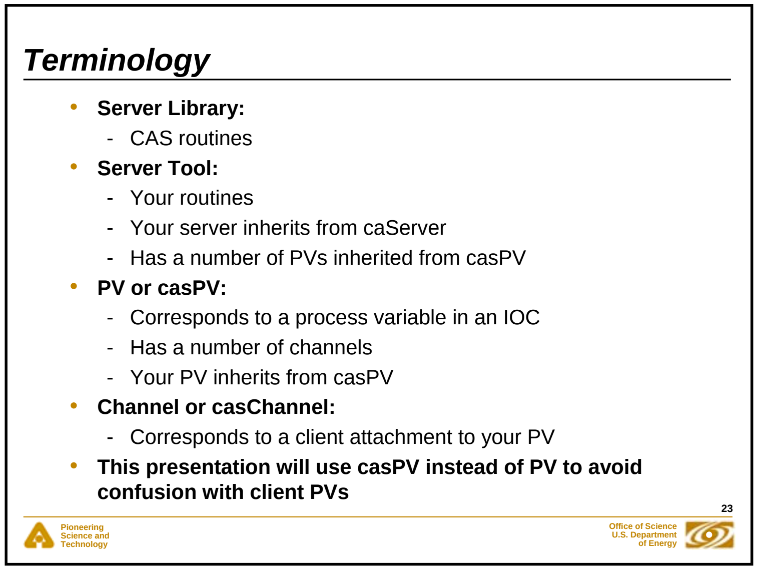


3.13 and 3.14 Differences

- **CAC**
  - Much effort has gone into making 3.13 clients work with 3.14
    - *You do not have to change your code**
  - CAC for 3.14 is threaded
- **CAS**
  - Channel Access Servers are significantly different in 3.14
    - *You have to change your code*
    - *Especially for timing routines*
      - osiTime vs. epicsTime
  - CAS is not threaded in either version

** Minor changes may be necessary depending on your coding*

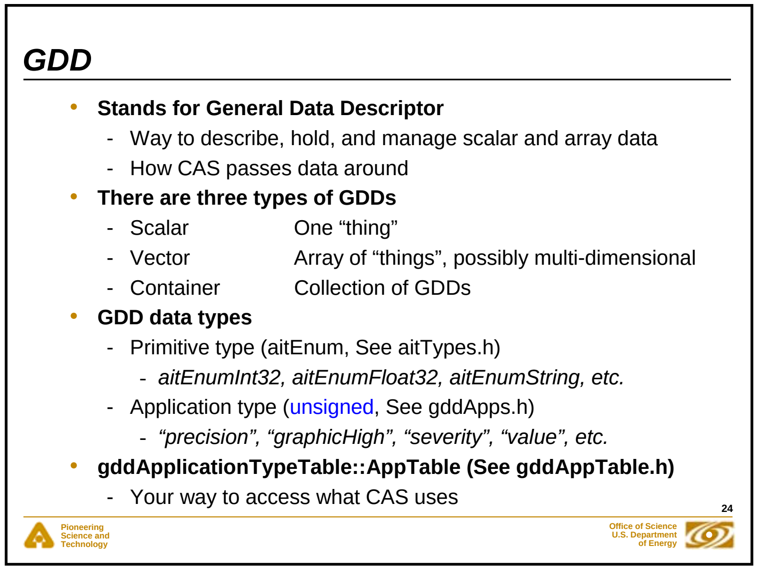
22



Terminology

- **Server Library:**
  - CAS routines
- **Server Tool:**
  - Your routines
  - Your server inherits from caServer
  - Has a number of PVs inherited from casPV
- **PV or casPV:**
  - Corresponds to a process variable in an IOC
  - Has a number of channels
  - Your PV inherits from casPV
- **Channel or casChannel:**
  - Corresponds to a client attachment to your PV
- **This presentation will use casPV instead of PV to avoid confusion with client PVs**

23



GDD

- **Stands for General Data Descriptor**
  - Way to describe, hold, and manage scalar and array data
  - How CAS passes data around
- **There are three types of GDDs**

Scalar	One “thing”
Vector	Array of “things”, possibly multi-dimensional
Container	Collection of GDDs

- **GDD data types**
  - Primitive type (aitEnum, See aitTypes.h)
    - *aitEnumInt32, aitEnumFloat32, aitEnumString, etc.*
  - Application type (unsigned, See gddApps.h)
    - *“precision”, “graphicHigh”, “severity”, “value”, etc.*
- **gddApplicationTypeTable::AppTable (See gddAppTable.h)**
  - Your way to access what CAS uses

24

GDD, cont'd

- **Reference counting**
  - GDDs are reference counted
  - When the count goes to zero, the destructor is called
  - Allows them to be passed around
- **A complete treatment of GDD would require a lecture in itself**
- **Best course is to look at some examples**
  - Keep your internal data in a GDD
  - Use gddApplicationTypeTable functions and smartGDDPointer
    - *getDD, smartCopy*
- **The reference manual for GDD is at**
  - http://www.aps.anl.gov/asd/controls/epics/EpicsDocumentation/EpicsGeneral/gdd.html
- **Also see**
  - epics/base/src/gdd

casdef.h

- **All C++ programs must include casdef.h**
  - `#include <casdef.h>`
- **You can look at this file to get more insight into CAS**
- **It is the instruction manual**

Basic Procedure for a Channel Access Server

- **Inherit from the appropriate CAS classes**
  - caServer
  - casPV
  - casChannel
- **Establish a loop to call fileDescriptionManager.process**
  - fileDescriptionManager.process(delay);
- **Respond to searches**
  - pvExistTest
- **Create PVs**
  - pvAttach (formerly createPV)
- **Respond to read, write, and other request from CAS**
- **Post events and other information to CAS**

- **CAS passes this all to and from the clients in your behalf**

fileDescriptorManager

- **CAS does its work in fileDescriptorManager.process**
  - Similarly to ca_pend in CAC
- **Typical main loop**

```
double delay=.01 // 10 ms
while(1) {
    // Let CAS do its stuff
    fileDescriptorManager.process(delay);
    // Do your stuff here
    ...
}
```

- **fileDescriptor is an external instance of fdManager**
  - fdManager::fileDescriptorManager;
  - Instantiated by CAS

caServer

```
class caServer {
    friend class casPVI;
public:
    caServer ();
    virtual ~caServer() = 0;

    virtual pvExistReturn pvExistTest (const casCtx & ctx,
      const caNetAddr & clientAddress,
      const char * pPVAliasName );
    virtual pvAttachReturn pvAttach (const casCtx &ctx,
      const char *pPVAliasName);
    casEventMask registerEvent (const char *pName);
    casEventMask valueEventMask () const; // DBE_VALUE
    casEventMask logEventMask () const;   // DBE_LOG
    casEventMask alarmEventMask () const; // DBE_ALARM
```

caServer

```
        void setDebugLevel (unsigned level);
        unsigned getDebugLevel () const;
        virtual void show (unsigned level) const;
        unsigned subscriptionEventsPosted () const;
        unsigned subscriptionEventsProcessed () const;
        class epicsTimer & createTimer ();
        void generateBeaconAnomaly ();
    private:
        class caServerI * pCAS;
        // Deprecated
        virtual class pvCreateReturn createPV (
          const casCtx & ctx, const char * pPVAliasName);
        virtual pvExistReturn pvExistTest (
          const casCtx & ctx, const char * pPVAliasName);
    };
```

pvExistTest

- **Called by CAS when it needs to determine if a PV exists**
  - Whenever it gets a search request packet
- **Is a virtual method**
- **Two overloaded versions**
  - New one also gives access to the caNetAddr
    - *Can be used to get the host name*
  - Old one is now deprecated
- **Three possible returns**

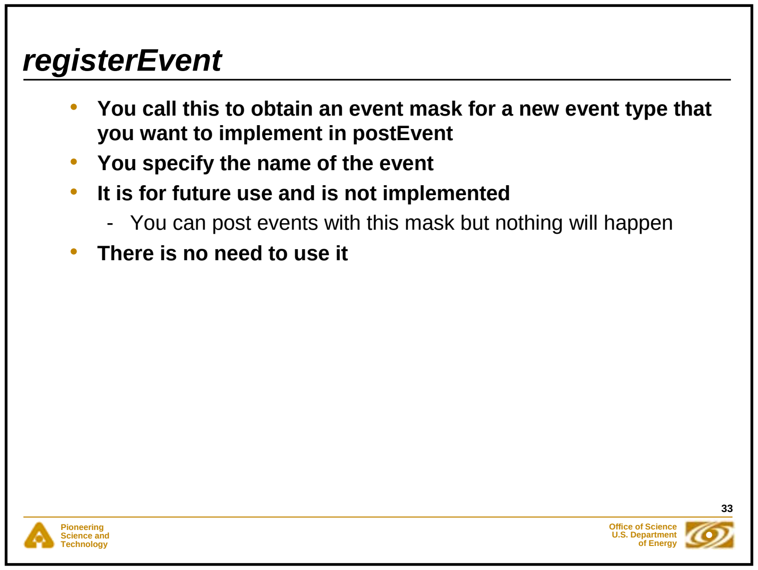
- pverExistsHere	(I have it)
- pverDoesNotExistHere	(I don't have it)
- pverAsynchCompletion	(I'll tell you later)

- **You can have aliases for your casPVs and respond to the aliases**

pvAttach

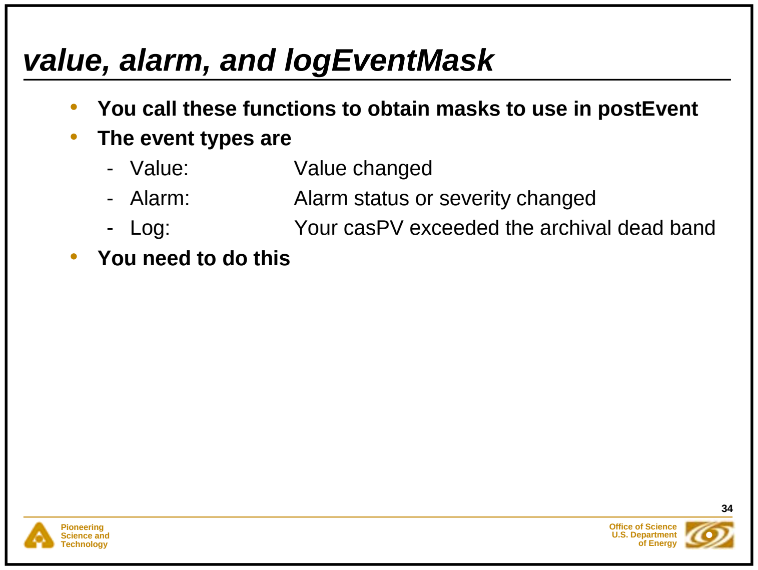
- **Called by CAS when a client attaches to the PV**
- **Used to be called createPV, now deprecated**
- **Is a virtual method**
- **You can create your casPV here or return it if it already exists**
- **Return value**
  - return casPV by pointer or reference

- S_casApp_pvNotFound	(No PV by that name here)
- S_casApp_noMemory	(Not enough resources)
- S_casApp_asyncCompletion	(I'll tell you later)



registerEvent

- **You call this to obtain an event mask for a new event type that you want to implement in postEvent**
- **You specify the name of the event**
- **It is for future use and is not implemented**
  - You can post events with this mask but nothing will happen
- **There is no need to use it**

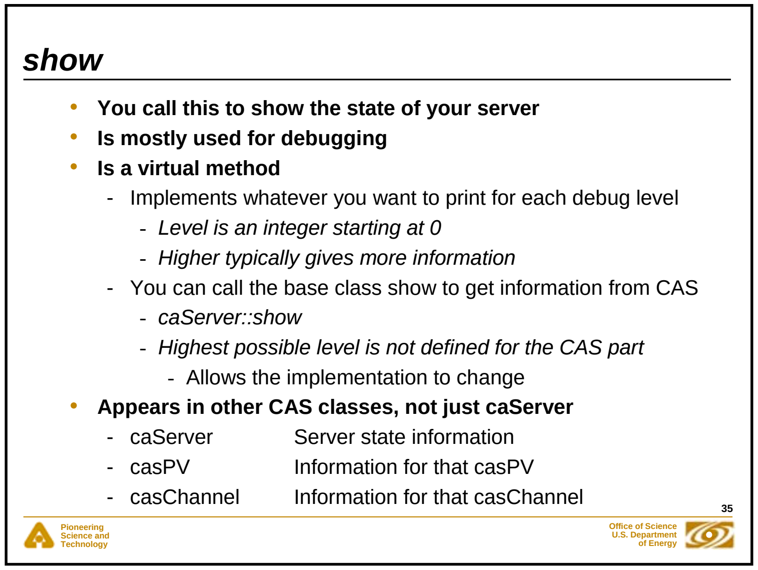


value, alarm, and logEventMask

- **You call these functions to obtain masks to use in postEvent**
- **The event types are**

- Value:	Value changed
- Alarm:	Alarm status or severity changed
- Log:	Your casPV exceeded the archival dead band

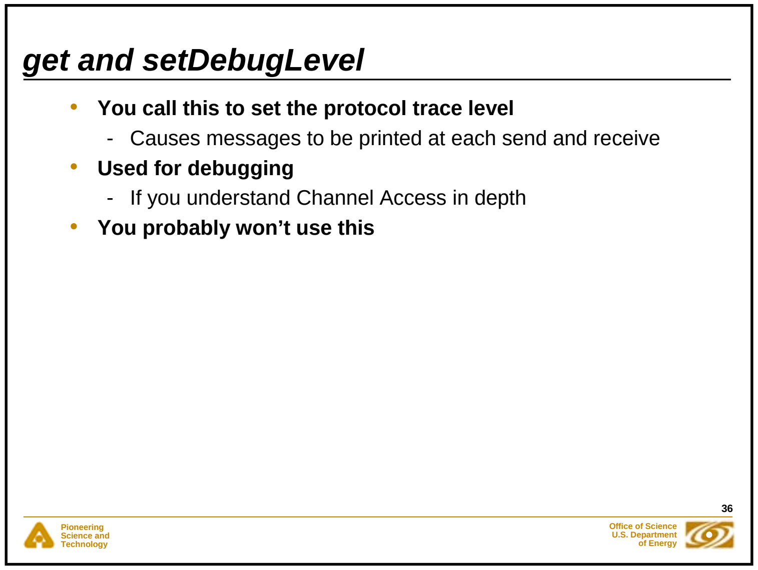
- **You need to do this**



show

- **You call this to show the state of your server**
- **Is mostly used for debugging**
- **Is a virtual method**
  - Implements whatever you want to print for each debug level
    - *Level is an integer starting at 0*
    - *Higher typically gives more information*
  - You can call the base class show to get information from CAS
    - *caServer::show*
    - *Highest possible level is not defined for the CAS part*
      - Allows the implementation to change
- **Appears in other CAS classes, not just caServer**

- caServer	Server state information
- casPV	Information for that casPV
- casChannel	Information for that casChannel



get and setDebugLevel

- **You call this to set the protocol trace level**
  - Causes messages to be printed at each send and receive
- **Used for debugging**
  - If you understand Channel Access in depth
- **You probably won't use this**

subscriptionEventsPosted and Processed

- You call these to get statistics on the number of events posted and processed
- They should be similar if you are keeping up
- Is optional
  - Used by the Gateway

createTimer

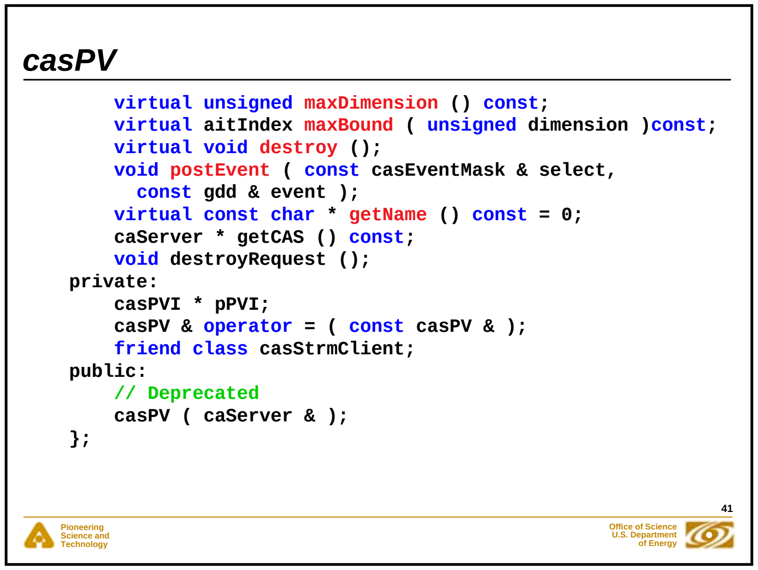
- You call this to create a timer using the same timer queue that CAS is using
- Is equivalent to calling fileDescriptorManager.createTimer
- You can also create your own timer queue
  - CAS queue is currently passive (not preemptive)
  - Yours can be active (preemptive)

generateBeaconAnomaly

- You call this if you want to generate extra beacon anomalies
  - Indicating a server coming up
- Normally you do not want to and should not do this
  - CAS does the usual beacon anomalies for you
- The Gateway does this when an IOC on its client side reconnects
  - To notify clients to retry their searches

casPV

```
class casPV {
public:
    casPV ();
    virtual ~casPV ();
    virtual void show ( unsigned level ) const;
    virtual caStatus interestRegister ();
    virtual void interestDelete ();
    virtual caStatus beginTransaction ();
    virtual void endTransaction ();
    virtual caStatus read (const casCtx &ctx,
      gdd &prototype);
    virtual caStatus write (const casCtx &ctx,
      const gdd &value);
    virtual casChannel * createChannel (
      const casCtx &ctx, const char * const pUserName,
      const char * const pHostName );
    virtual aitEnum bestExternalType () const;
```



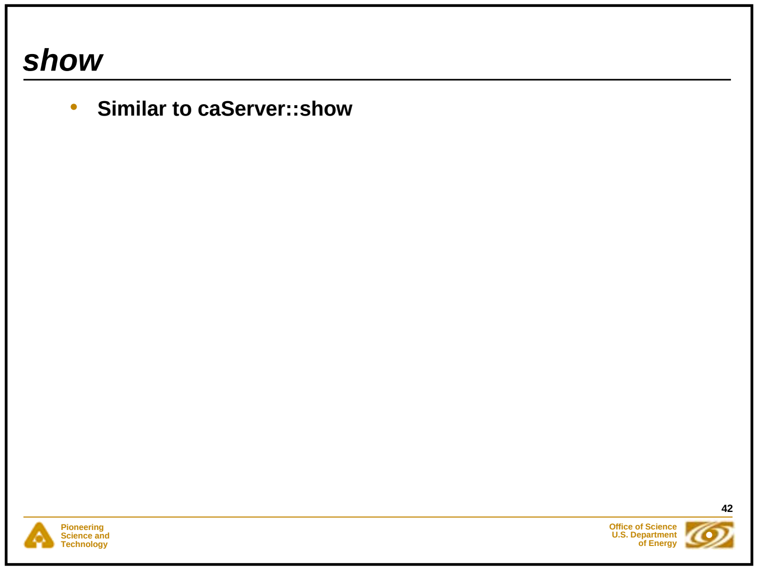
casPV

```
        virtual unsigned maxDimension () const;
        virtual aitIndex maxBound ( unsigned dimension )const;
        virtual void destroy ();
        void postEvent ( const casEventMask & select,
          const gdd & event );
        virtual const char * getName () const = 0;
        caServer * getCAS () const;
        void destroyRequest ();
    private:
        casPVI * pPVI;
        casPV & operator = ( const casPV & );
        friend class casStrmClient;
    public:
        // Deprecated
        casPV ( caServer & );
    };
```

41

Pioneering Science and Technology

Office of Science U.S. Department of Energy



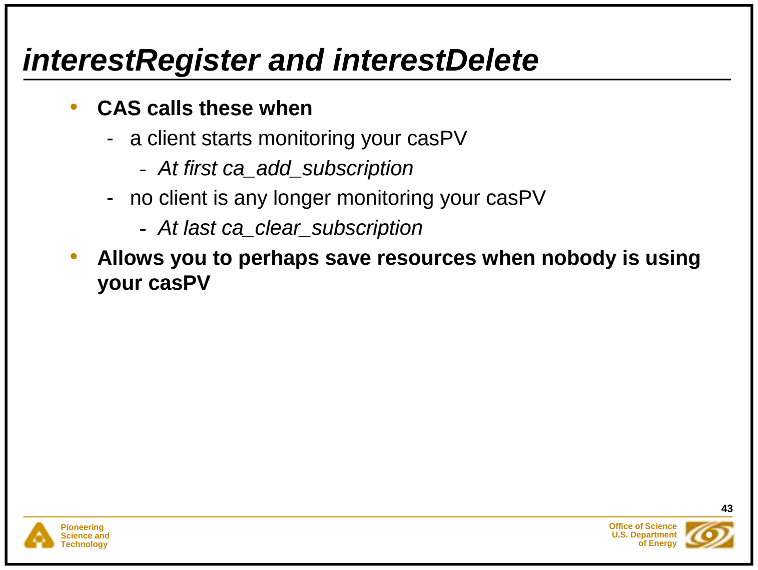
show

- Similar to caServer::show

42

Pioneering Science and Technology

Office of Science U.S. Department of Energy



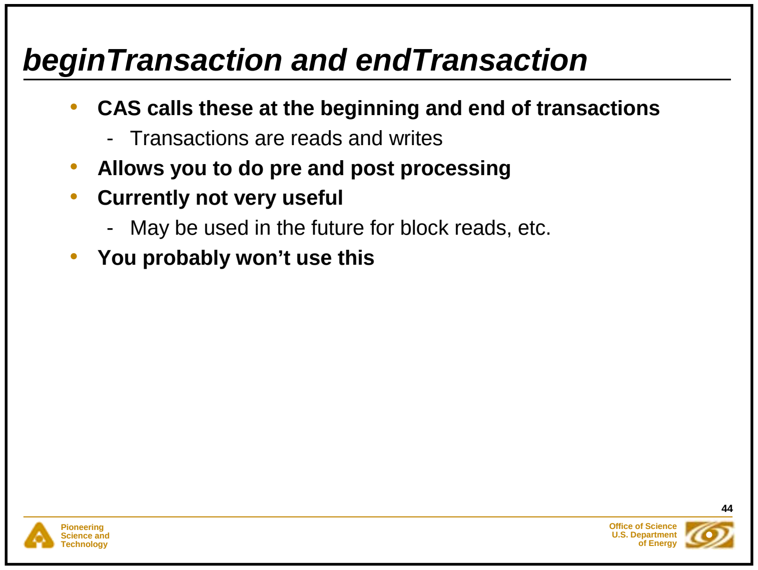
interestRegister and interestDelete

- **CAS calls these when**
  - a client starts monitoring your casPV
    - *At first ca_add_subscription*
  - no client is any longer monitoring your casPV
    - *At last ca_clear_subscription*
- **Allows you to perhaps save resources when nobody is using your casPV**

43

Pioneering Science and Technology

Office of Science U.S. Department of Energy



beginTransaction and endTransaction

- **CAS calls these at the beginning and end of transactions**
  - Transactions are reads and writes
- **Allows you to do pre and post processing**
- **Currently not very useful**
  - May be used in the future for block reads, etc.
- **You probably won't use this**

44

Pioneering Science and Technology

Office of Science U.S. Department of Energy

read and write

- **Called by CAS when it want to read or write your casPV**
  - Which is when a client wants to
- **You fill in the GDD or read from it and return the status**
- **Return values**

- S_casApp_success	(OK)
- S_casApp_asyncCompletion	(I'll do it later)
- S_casApp_postponeAsyncIO	(You do it later)
- S_casApp_noSupport	(I don't handle that)
- S_casApp_noMemory	(I'm in trouble)
- Others in casdef.h	

createChannel

- **Called by CAS whenever a client attaches to a PV**
- **You don't have to implement it unless you are keeping channel information**
- **You probably don't need channel information unless you are implementing access security**

bestExternalType

- **You return the "native" type of your casPV**

```
typedef enum {          // See aitTypes.h
  aitEnumInvalid=0,
  aitEnumInt8,
  aitEnumUint8,
  aitEnumInt16,
  aitEnumUint16,
  aitEnumEnum16,
  aitEnumInt32,
  aitEnumUint32,
  aitEnumFloat32,
  aitEnumFloat64,
  aitEnumFixedString,
  aitEnumString,      // This is the default
  aitEnumContainer } aitEnum;
```

maxDimension and maxBound

- **You return an unsigned int for maxDimension and an aitIndex (= aitUint32 = unsigned int) for maxBound**
  - Scalar: 0
  - Array: 1
    - *maxBounds(0) supplies number of elements in array*
  - Plane: 2
    - *maxBounds(0) supplies number of elements in X dimension*
    - *maxBounds(1) supplies number of elements in Y dimension*
  - Cube : 3
    - *maxBounds(0) supplies number of elements in X dimension*
    - *maxBounds(1) supplies number of elements in Y dimension*
    - *maxBounds(2) supplies number of elements in Z dimension*
- **The default maxDimension returns 0 (scalar) and the default maxBound returns 1 (scalar bounds) for all dimensions**

destroy

- **Called by CAS**
  - When last client becomes unattached
  - For each casPV when the server is deleted
- **Is a virtual method**
  - You can do whatever you want
- **If you don’t implement it, CAS calls your destructor**
- **Allows you to clean up**
- **Note that your casChannels are destroyed before your casPV**

postEvent

- **You call this when you want to post an event**
  - Owing to changes in your casPV
  - You pass a GDD by reference

getName

- **CAS calls this to ask for the name of your casPV**
- **CAS does not store your name**
  - You have responsibility for the name
- **The pointer must remain valid for the life of the casPV**
- **You should return the base name when there are aliases**

getCAS

- **You call this when you want to get the caServer associated with this channel**
- **Is a convenience function**

destroyRequest

- CAS calls this when it is ready to destroy your casPV
- Afterward
  - It calls your virtual destroy method
  - It unlinks your casPV from the library
- Do not call it yourself
- There seems to be no reason to use this

casChannel

```
class casChannel {
public:
    casChannel ( const casCtx & ctx );
    virtual ~casChannel ();
    virtual void setOwner ( const char * const pUserName,
      const char * const pHostName );
    virtual bool readAccess () const;
    virtual bool writeAccess () const;
    virtual bool confirmationRequested () const;
    virtual caStatus beginTransaction ();
    virtual void endTransaction ();
    virtual caStatus read (const casCtx &ctx,
      gdd &prototype);
    virtual caStatus write (const casCtx &ctx,
      const gdd &value);
    virtual void show ( unsigned level ) const;
    virtual void destroy ();
```

casChannel

```
       void postAccessRightsEvent ();
       casPV * getPV ();
       void destroyRequest ();
    private:
        class casChannelI * pChanI;
        casChannel ( const casChannel & );
        casChannel & operator = ( const casChannel & );
        friend class casStrmClient;
    };
```

setOwner

- You call this when you want change the user or host name
- Not usually done

readAccess and writeAccess

- **Called by CAS to determine access before each read and write**
- **Used if you implement access security**
- **Access security is done at the casChannel level**
  - Since it depends on the client

confirmationRequested

- **You call this**
- **It is designed to implement another type of write request**
  - Where the client has to confirm before writing
  - Example:
    - *“Setting this will shut down the whole project. Are you sure?”*
- **It is not implemented**
- **You should not use it**

beginTransaction and endTransaction

- **Similar to casPV::beginTransaction and endTransaction**
- **CAS calls these at the beginning and end of transactions**
  - Transactions are reads and writes
- **Allows you to do pre and post processing**
- **Currently not very useful**
  - May be used in the future for block reads, etc.
- **You probably won’t use this**

read and write

- **Called by CAS when it want to read or write your casChannel**
  - Which is when a client wants to
- **If not implemented, it calls casChannel::read or casChannel::write**
- **You usually do not implement these for the casChannel**
  - Do it for the casPV

show

- **Similar to caServer::show**

destroy

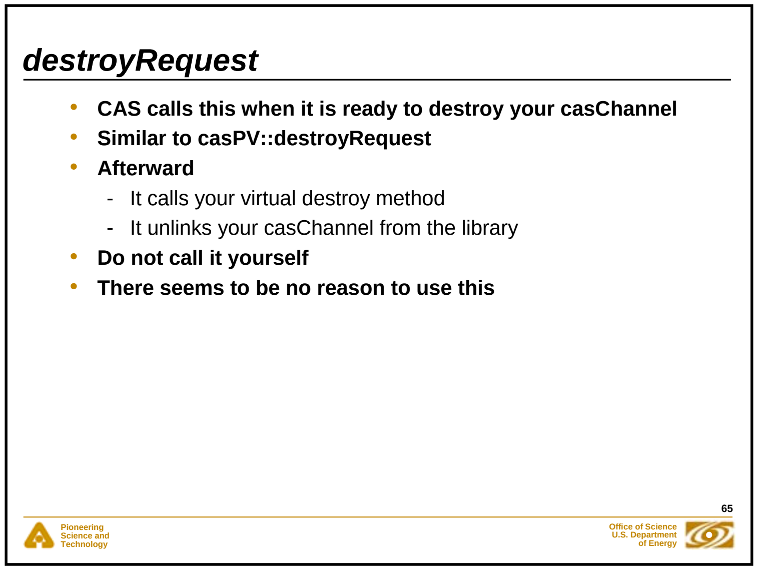
- **Called by CAS**
  - When the client becomes unattached
- **Similar to casPV::destroy**
- **Is a virtual method**
  - You can do whatever you want
- **If you don’t implement it, CAS calls your destructor**
- **Allows you to clean up**
- **Note that your casChannels are destroyed before your casPV**

postAccessRightsEvent

- **You call this when you want to change the access rights**
- **Used if you implement access security**

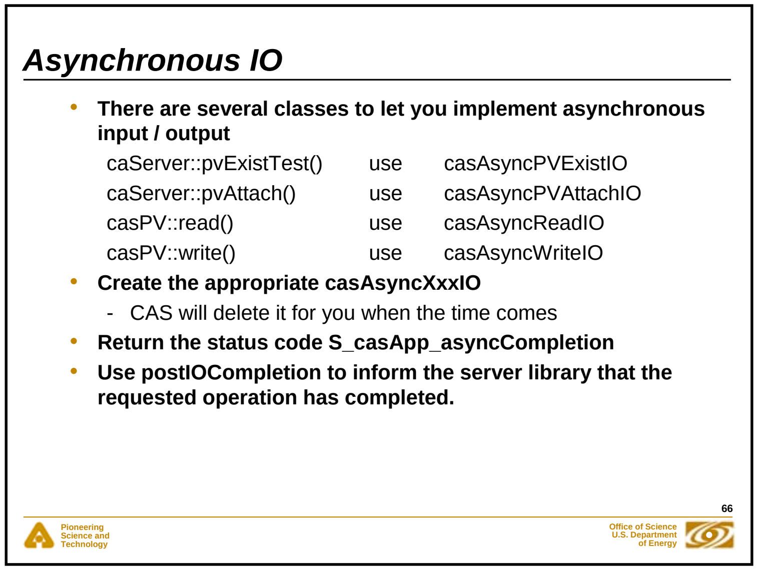
getPV

- **You call this when you want to get the casPV associated with this channel**
- **Is a convenience function**



destroyRequest

- CAS calls this when it is ready to destroy your casChannel
- Similar to casPV::destroyRequest
- Afterward
  - It calls your virtual destroy method
  - It unlinks your casChannel from the library
- Do not call it yourself
- There seems to be no reason to use this

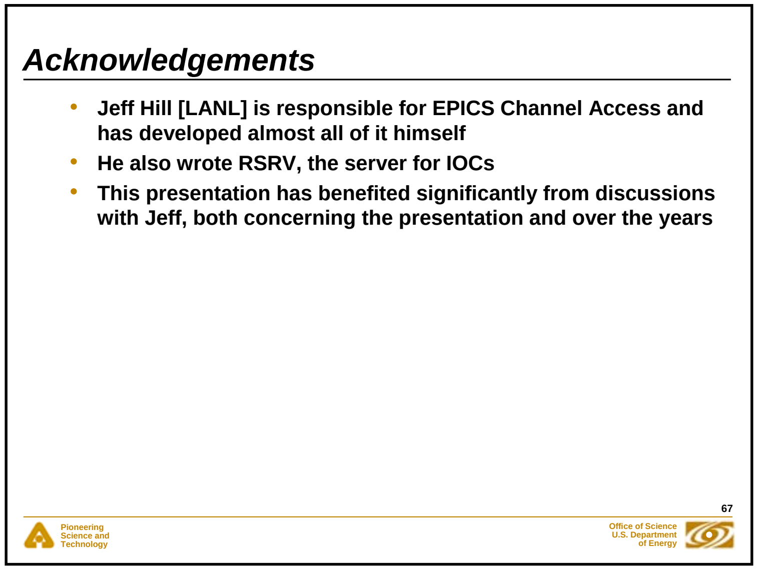


Asynchronous IO

- There are several classes to let you implement asynchronous input / output

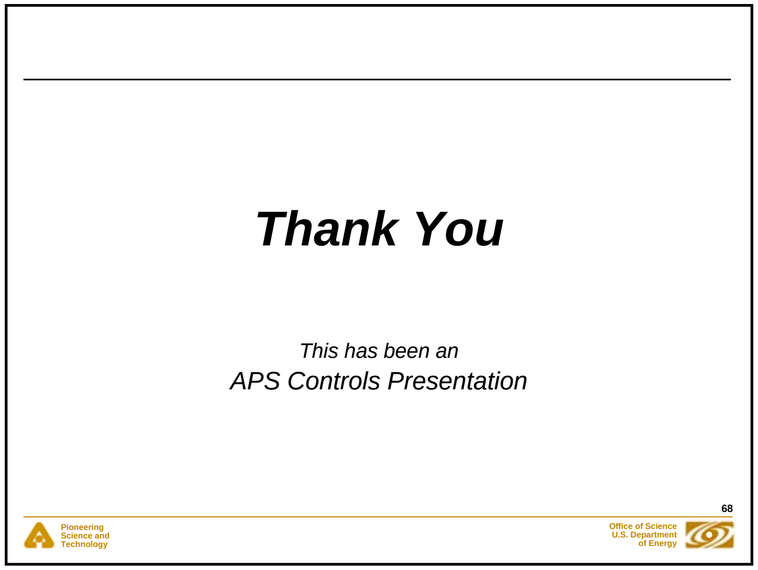
caServer::pvExistTest()	use	casAsyncPVExistIO
caServer::pvAttach()	use	casAsyncPVAttachIO
casPV::read()	use	casAsyncReadIO
casPV::write()	use	casAsyncWriteIO

- Create the appropriate casAsyncXxxIO
  - CAS will delete it for you when the time comes
- Return the status code S_casApp_asyncCompletion
- Use postIOCompletion to inform the server library that the requested operation has completed.



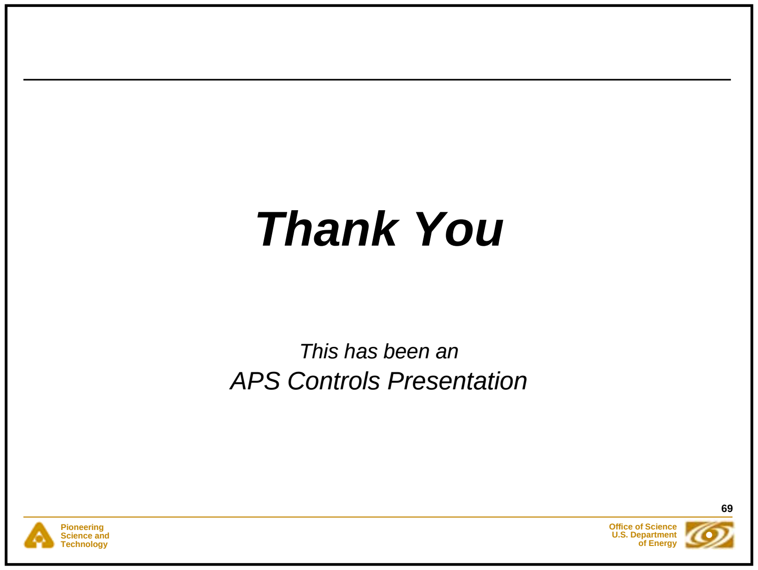
Acknowledgements

- Jeff Hill [LANL] is responsible for EPICS Channel Access and has developed almost all of it himself
- He also wrote RSRV, the server for IOCs
- This presentation has benefited significantly from discussions with Jeff, both concerning the presentation and over the years



Thank You

This has been an
APS Controls Presentation



Thank You

This has been an

APS Controls Presentation

69

Pioneering Science and Technology

Office of Science U.S. Department of Energy