Portable Channel Access Server

Marty Kraimer
Overview

◆ What is the Portable Channel Access Server?

The Portable Server consists of a C++ library with a simple class interface.

◆ Purpose of the Server library

Using the simple interface to the library, a developer can create a Channel Access server tool that can interact with the EPICS database as well as other applications.

◆ Example ca servers
  - Channel access gateway
  - Directory server
  - Fault logger APT HPRF
  - KECK instruments
  - KEKB gateway to LINAC control system
  - SLAC gateway to SLAC control system
  - Gateways to other control systems at DESY
Advantages of a Server Tool

- Your application becomes an EPICS server tool
- Your data become EPICS process variables
- MEDM and other EPICS tools can interact with your application

Talk purpose

- Describe the server interface
- Show simple C++ example server
Basic Concepts

Server Tool
- Developer creates a channel access server tool
- Provides interface classes and methods required by the server library

Server Tool Functions
- Creates/deletes server instance
- Responds to client requests
  - PV search
  - Attach/detach request
  - Read/write requests
- Posts change of state events

Server Library
- C++ library with simple class interface
- Calls the C++ server interface functions
- Developer only needs to know server interface
- Hides complexity of channel access
- Available in EPICS base
- Runs on Unix, WIN32, and VMS
Basic Concepts (cont.)

❖ Process Variable (PV)
  ❖ Variable which server tool keeps track of
  ❖ Server tool provides clients with current value when requested (read request)
  ❖ Server tool changes current value upon client request (write request)
  ❖ Server tool can inform client when the current value changes (monitoring)
  ❖ Has attributes (e.g. alarm limits, operating range) which server tool keeps track of

❖ Channel
  ❖ A connection between a client and a PV
  ❖ Each client establishes a separate connection to the PV
9 classes comprise the Portable Server API

- Server class, caServer
- Process variable class, casPV
- pvExistReturn
- pvAttachReturn
- Channel class, casChannel
- casAsyncPVExistIO
- casAsyncCreatePVIO
- casAsyncReadIO
- casAsyncWriteIO.

- The first four classes are required to implement the server tool
- The channel class and the asynchronous IO classes can be used to add more functionality
- Each class has several member functions which server tool must define
caServer Class

- Every server tool must include a class derived from the caServer class
- Defines maximum length of a PV name
- Defines debug level determining amount of output printed
- Determines maximum number of simultaneous IO operations allowed
- Informs the server library if a PV is associated with the server tool
- Attaches a PV when a client wishes to establish a connection
- Server tool must provide implementations of the virtual functions
  - pvExistTest()
  - pvAttach()
Example Server

◆ Server definition

class myServer : public caServer
{
public:
    myServer(unsigned pvCountIn, char *nameIn);
    virtual ~myServer(void);
    virtual pvExistReturn pvExistTest(const casCtx& c, const char* pvname);
    virtual pvAttachReturn pvAttach(const casCtx& c, const char* pvname);
private:
    friend class myPV;
    myPV *mypv;
    char *pvName;
    int pvNameLength;
    gdd* value;
};

◆ Server creation

int main(int argc, char* argv[]){
    myServer* server;
    int forever=1;

    if(argc<2) {
        fprintf(stderr,"Usage: %s pvName\n",argv[0]);
        return -1;
    }
    server = new myServer(1,argv[1]);
    osiTime delay(1000u,0u);
    while(forever) {
        fileDescriptorManager.process(delay);
    }
    return 0;
}
pvExistTest

pvExistReturn pvExistTest(const casCtx &ctx, const char *pPVAliasName)

- Response to a client CA search
- The server tool may accept multiple PV name aliases for the same PV.
- The request is allowed to complete asynchronously (server tool uses asynchronous IO classes).
- Server tool passes ctx to asynchronous completion constructors
- Return values (class pvExistReturn)
  - return pverExistsHere;
    Server has PV
  - return pverDoesNotExistHere;
    Server does not know of this PV
  - return pverAsynchCompletion;
    Deferred result
pvAttach

pvAttachReturn pvAttach (const casCtx &ctx, const char *pPVAliasName)

- Called when client wishes to attach to PV
- Allowed to complete asynchronously
- Server tool must detect attempts to create a 2nd PV with the same name
- Return values (class pvAttachReturn)
  - return pPV;
    Success (pass by pointer)
  - return PV;
    Success (pass by ref)
  - return S_casApp_pvNotFound;
    No PV by that name here
  - return S_casApp_noMemory;
    No resources to create pv
  - return S_casApp_asyncCompletion;
    Deferred completion
  - return S_casApp_postponeAsyncIO;
    Too many simultaneous IO operations
Example Server Methods

myServer::myServer(unsigned pvCountIn,char *nameIn)
{
    pvNameLength = strlen(nameIn);
    pvName = new char [pvNameLength+1];
    strcpy(pvName,nameIn);
    value = new gddScalar(appvalue,aitEnumFloat64);
    value->reference();
    value->put(0);
    value->setStatSevr(0,0);
    mypv = new myPV(*this,pvName);
}

pvExistReturn myServer::pvExistTest(const casCtx&, const char* name)
{
    if(strncmp(name,pvName,pvNameLength)==0)
        return pverExistsHere;
    return pverDoesNotExistHere;
}

pvAttachReturn myServer::pvAttach(const casCtx&,const char* name)
{
    if(strncmp(name,pvName,pvNameLength)==0) return *mypv;
    return NULL;
}

myserver::~myserver(void)
{
    delete [] pvName;
    value->unreference();
    delete mypv;
}
casPV Class

- Responds to read/write PV requests
  - Server must implement the virtual functions
    - read()
    - write()

- Responds to a request for a PV monitor
  - Server implements the virtual functions
    - interestRegister()
    - interestDelete()
  - Calls postEvent()

- Other important functions
  - getName()
  - bestExternalType()
  - beginTransaction(), endTransaction()
  - destroy()

- Do nothing default implementations exist.
- Server tool need not implement those functions it does not want.
Example casPV
Class Definition

class myPV : public casPV
{
public:
    myPV(myServer& serverIn,char* nameIn);
    virtual ~myPV(void);
    virtual void destroy(void);
    virtual caStatus read(const casCtx &, gdd &prototype);
    virtual caStatus write(const casCtx &, gdd &dd);
    virtual aitEnum bestExternalType(void) const;
    virtual caStatus interestRegister(void);
    virtual void interestDelete(void);
    virtual const char *getName() const;
private:
    myServer& server;
    char *pvName;
    int interest;
};
Example casPV Methods

myPV::myPV (myServer& svrIN, char * nameIn):
    server(svrIN), interest(0)
{
    pvName = new char [strlen(nameIn)+1];
    strcpy(pvName, nameIn);
}

casStatus myPV::read(const casCtx&, gdd &dd)
{
    dd.put(server.value);
    return S_casApp_success;
}

casStatus myPV::write(const casCtx&, gdd &dd)
{
    aitFloat64 newValue;
    dd.get(&newValue, aitEnumFloat64);
    server.value->put(newValue);
    if (interest) postEvent(server.valueEventMask,*value);
    return S_casApp_success;
}
Example casPV Methods (cont.)

```cpp
aitEnum myPV::interestRegister(void)
{
    interest = 1;
    return S_casApp_success;
}

void myPV::interestDelete(void) { interest = 0; }

const char *myPV::getName() const { return pvName; }

aitEnum myPV::bestExternalType() const
{
    return aitEnumFloat64;
}

myPV::~myPV(void) { delete [] pvName; }

void myPV::destroy(void) { }
```
Data Types

- Channel Access client request types
  DBR types defined in db_access.h
  e.g. DBR_STS_CHAR, DBR_GR_DOUBLE

- EPICS database native types
  DBF types defined in db_access.h
  e.g. DBF_DOUBLE, DBF_STRING,...

- Server has two types which describe data
  - Architecture Independent Type (AIT) defined in aitTypes.h
    - aitInt8    aitUint8    aitInt16
    - aitUint16  aitFloat32  aitFloat64
    - aitEnum16  aitIndex    aitPointer
    - aitStatus
  - Application type defined in gddAppTable.h
    - e.g. precision, limits, status

- GDD library converts data from one type to another
Writing Your Own Server Tool

- Next steps
  - Try existing samples
  - Study sample code
  - Study casdef.h
  - Read documentation
On-line documents at LANL

- Portable Server Tutorial
- Portable Server Reference
- A Server-Level API for EPICS (paper)
- Channel Access Server Update (slides)