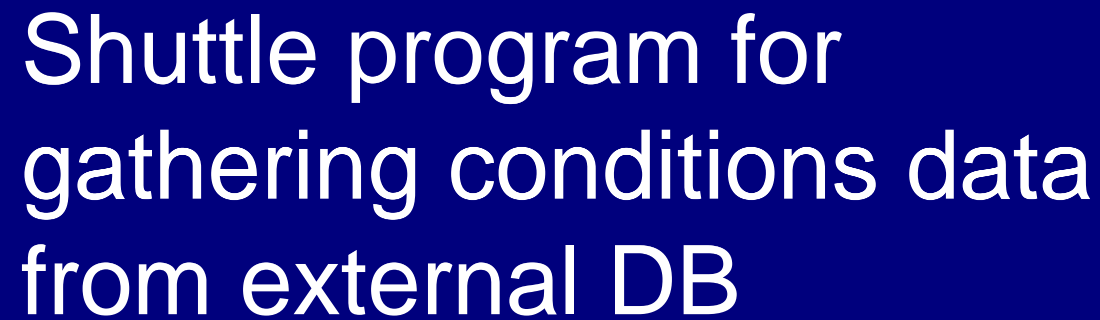


Shuttle program for gathering conditions data from external DB

Boyko Yordanov

4 October 2005

ALICE Offline week

Outline

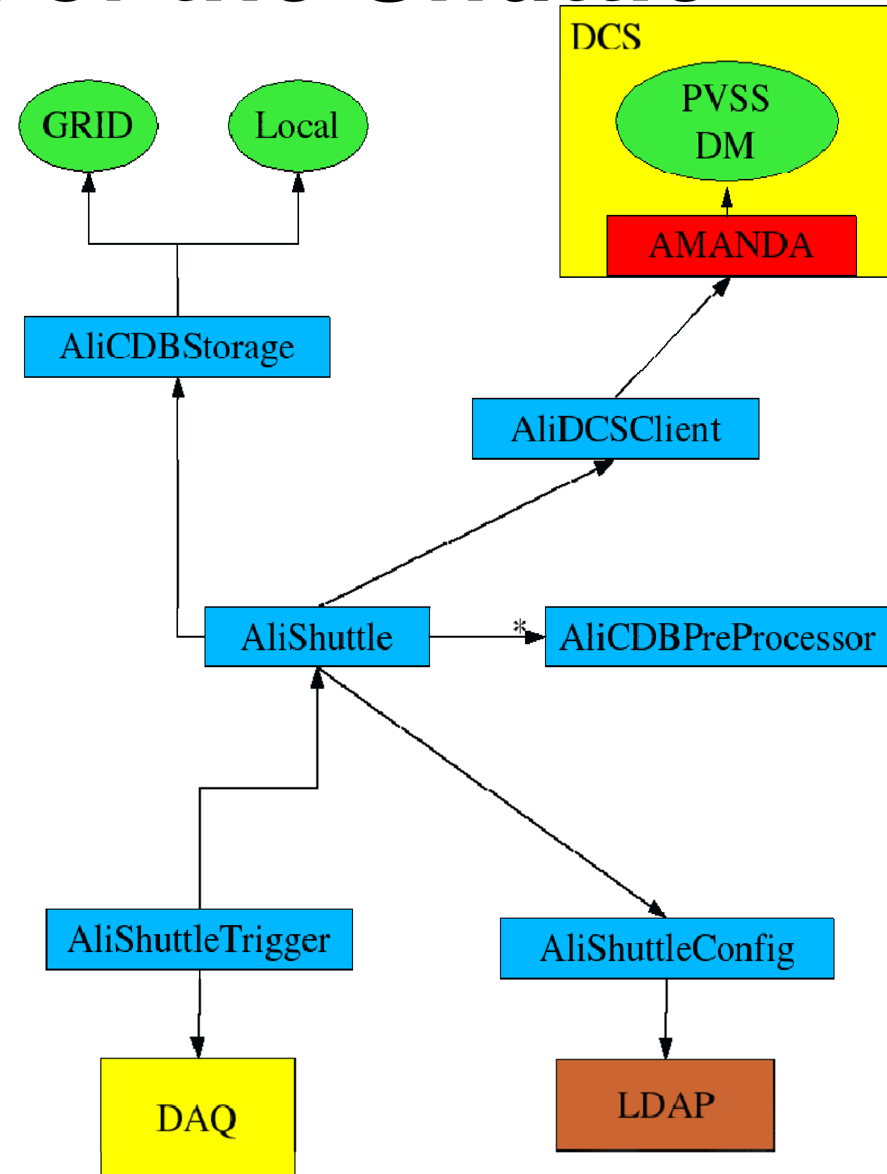
- General Overview
- DCS Conditions Data Model
- CDB Preprocessors and Default Storage
- Shuttle Configuration
- DCS API
- General Tests
- Conclusion

- Collecting conditions data from external DCS DB at predetermined time intervals

  (for example at the end of every run)
- Processing collected data using specific CDB preprocessor for every sub-detector
- Storing conditions data in the Conditions Data Base (CDB)

General Schema of the Shuttle

- *AliShuttle* – The Shuttle program manager. Organizes conditions data retrieval, preprocessing and storing it into CDB.
- *AliShuttleConfig* – Interface to the configuration stored into LDAP server
- *AliCDBStorage* – CDB storage interface
- *AliDCSClient* – Provides DCS API. Communicates with DCS ***AMANDA*** server over TCP/IP



Conditions Data in DCS DB

- The abstract representation of every conditions parameter (temperature, voltage etc.) is organized in DataPoints (DP)
- Additional attribute "*alias*" is assigned to every DP to provide constant identifier for the relevant parameter
- Conditions data is organized in *value/timestamp* series (one for every parameter). As the DCS data is collected independently on the experimental runs, part of it doesn't belong to any run
- Every DP representing conditions parameter has one of the following types:

Primitive type	Dynimic type
Boolean	DynBoolean
Byte	DynByte
Integer	DynInteger
UInteger	DynUInteger
Float	DynFloat

DCS Conditions Data in AliRoot

- Particular value/timestamp pair is organized in *AliDCSValue* object with the following composite structure:
  - *AliSimpleValue* - Union like holder for DP value. Provides setter and getter for every DP type
  - *UInt_t* - Timestamp field
- The value/timestamp series is represented by collection (*TList*) of *AliDCSValue* objects
- Two *value/timestamp* (before and after requested interval) pairs are included into the series (if they exist). Especially useful if *extrapolation/interpolation* procedures are used
- As *AliDCSValue* is a ROOT object that can be directly stored into the Conditions DB

Calibration Data Preprocessors

- Motivation for a CDB preprocessor:
  - Assuming that the DCS data is *slowly changing* and *well-behaved* over a “long” time period:
    - User-defined *objects* can be created (histograms, functions, average values) depending on the data type and observed behavior
    - The amount of data stored in CDB will be minimized
    - It is simpler to use specific object than long value/timestamp series
- Every sub-detector can use its own CDB preprocessor implementing *AliCDBPreProcessor* interface:
  - Allows for intelligent treatment of raw conditions data
- If specific preprocessor is not provided, the raw series *value/timestamp* is stored (default behavior)

AliCDBPreProcessor

- *AliCDBPreProcessor* is a subclass of *TNamed*. Method *TNamed::GetName()* is used as preprocessor identifier (sub-detector name)

- *AliShuttle::RegisterCDBPreProcessor(AliCDBPreProcessor*)* is used to register specific preprocessor to the Shuttle manager

- *AliCDBPreProcessor* interface methods:
  - *void Initialize(Int_t run, UInt_t startTime, UInt_t endTime)* – Called at the beginning of conditions data retrieval (Before the first alias is processed)
  - *void Finalize()* – Called at the end of conditions data retrieval (After the last alias is processed)
  - *void Process(const char* alias, TList& valueSet, Bool_t hasError)* – Called sequentially for every alias in the configuration after its data is retrieved from DCS DB

- *AliCDBPreProcessor* helper methods:
  - *Bool_t Store(const char* specType, TObject* object, AliCDBMetaData* metaData)* – Stores object and metaData to the underlying CDB storage using pathname: <detector>/DCS/<sepcType>

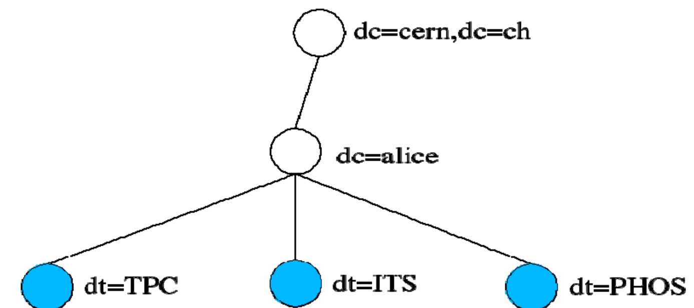
Default Storage to CDB

- Whole *value/timestamp* series (*TList* of *AliDCSValue* object) is stored to CDB with pathname: <detector>/DCS/<alias>
- Two properties are added to *AliCDBMetaData*:
  - StartTime - *AliSimpleValue* denoting run start time
  - EndTime - *AliSimpleValue* denoting run end time
- Getting run start time example:

```
AliSimpleValue* startTimeHolder;
startTimeHolder = (AliSimpleValue*) metaData->GetProperty("StartTime");
UInt_t startTime = startTimeHolder->GetUInt();
```

Shuttle Configuration

- Conditions data is retrieved only for those detectors for which there is a valid configuration
- LDAP server is used for keeping configuration entries and organizing access policy
- *AliShuttleConfig* provides transparent interface to the underlying configuration server
- TLDAP package in ROOT is used as API for the LDAP server
  - *ipHost* – Single value attribute describing AMANDA server host
  - *ipServicePort* – Single value attribute describing AMANDA server port
  - *alias* – Multi value parameter describing the set of aliases which will be retrieved from DCS DB



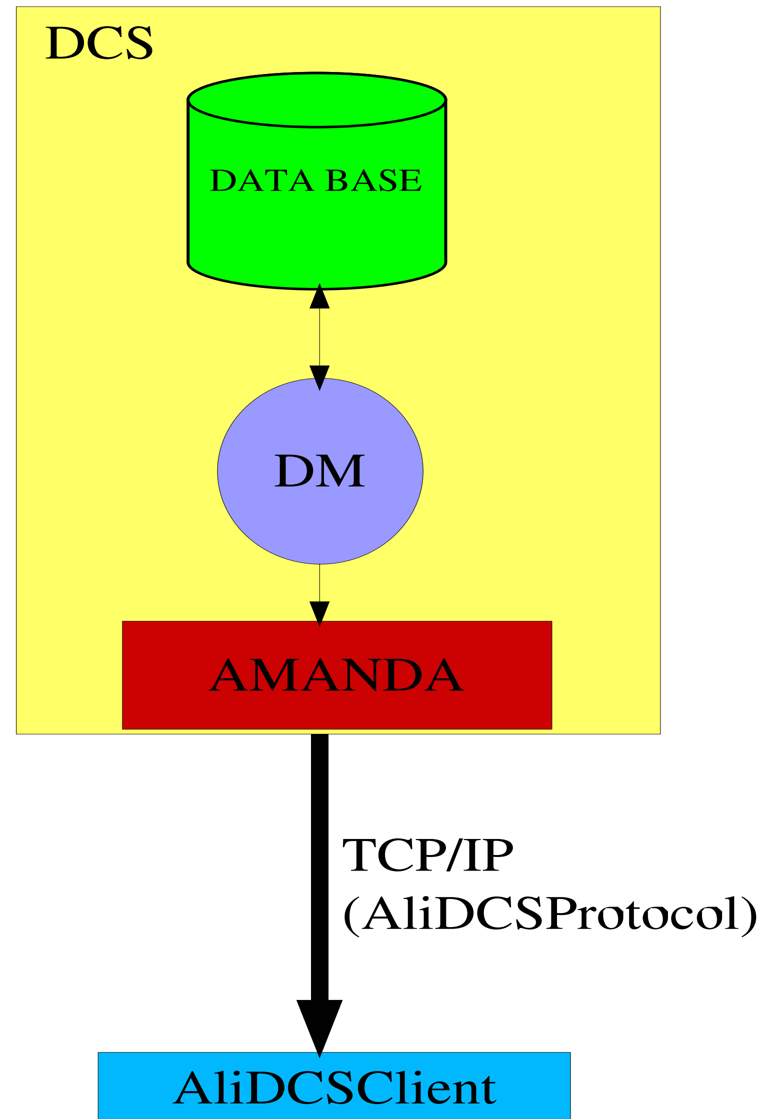
Configuration Example

- Example of sub-detector entry in ldif format which can be written to LDAP server

```
#TPC config
dn: dt=TPC,dc=alice,dc=cern,dc=ch
objectClass: shuttleConfig
dt: TPC
ipHost: 192.168.39.21
ipServicePort: 4242
alias: HighVol01
alias: HighVol02
alias: TpcTempSect01
alias: TpcTempSect02
```

AliDCSClient – DCS API

- *DM* – Data manager part of PVSS SCADA system which organizes the work with the underlying data base.
- *AMANDA* – Communication layer implementing server side of AliDCSProtocol
- *AliDCSClient* – Client side of AliDCSProtocol which provides DCS API



AliDCSClient Overview

- *Int_t GetDPValues(const char* dpName, UInt_t startTime, UInt_t endTime, TList& result)* - Retrieves data from DCS DB for data point *dpName* in time interval *startTime – endTime*

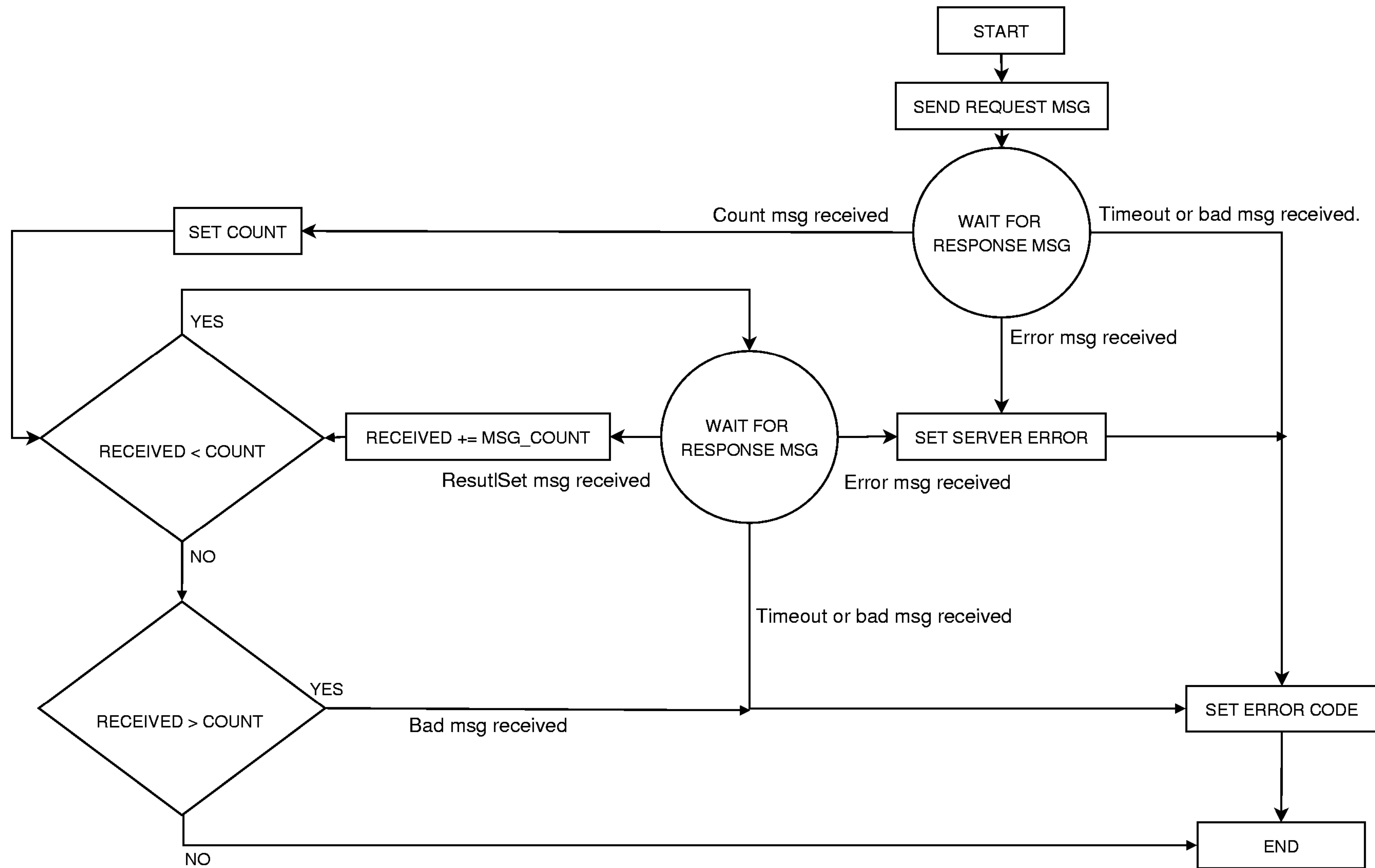
- *Int_t GetAliasValues(const char* alias, UInt_t startTime, UInt_t endTime, TList& result)* - Retrieves data from DCS DB for *alias* in time interval *startTime – endTime*

- In case of negative value returned it indicates that some of the following errors occurred:
  - BadState
  - Timeout
  - BadMessage
  - CommunicationError
  - ServerError

AliDCSProtocol

- AliDCSProtocol is simple message protocol over TCP/IP
- Every message has header with fixed structure and variable body depending on the message type
- Following message types constitute the protocol:
  - ***Request*** – Sent by AliDCSClient to initiate data retrieval
  - ***Count*** – Sent by Amanda server to indicate total number of *value/timestamp* pairs belonging to the request
  - ***ResultSet*** – Contains part of *value/timestamp* series returned by Amanda server
  - ***Error*** – Sent by Amanda server in case of error

AliDCSProtocol Flow Chart



General Tests

- Tests of DM-AMANDA -> AliDCSClient have been discussed with DCS team:
  - **DCS DB**: 20GB test data generated with about 30 data points (2 Hz update data frequency for every DP series)
  - **Client**: Sequential data retrieval for 1 DP and time interval 1 hour (about 7200 values per series)
  - The amount of data requested will be increased gradually (changing the number of DP and increasing the time interval)
  - The client side test suite ready. Tests will be done with a stable version of AMANDA server (also ready), provided by the DCS group

Conclusion

Features and benefits of the presented approach:

- Avoiding high load on the external DB: periodic polling, long series of data extracted with one call and small overhead
- External DBs are accessed only for data *relevant* to offline processing/analysis
- Only this data is stored in the Conditions DB
- AliCDBPreProcessor allows for the raw calibration/alignment data to be treated before storing into the Conditions DB: *smart treatment*, minimizes the amount of calibration/alignment data in memory during reconstruction
- *Unified method* for accessing replicated data is provided by CDB storage infrastructure
- *Worldwide availability* through the Grid