# The Hall D Online Component Database


# Elliott Wolin


# 5-Mar-2012

## Introduction

Below I list preliminary requirements for a Hall D component database, followed by results of an initial survey and recommendations for how to proceed.

Along with the usual inventory information commonly maintained, we further would like to keep information on component power and control requirements, repair history, etc.

We have both near- and long-term requirements. In the near term we would like to deploy an inventory system immediately to track receipt, testing and installation of equipment in the hall and test areas. In the longer term we would like to eventually capture all information useful for managing equipment in the experiment. Thus we may start with one system and migrate to a better, more comprehensive system at a later date. Fortunately it is generally easy to move data from one relational database to another, even if the underlying schemas are somewhat different.

Different sub-groups in Hall D have different needs and requirements for a component/inventory database system. Some have such simple requirements that spreadsheets suffice, and they are not interested in using a more sophisticated system. We will consider importing their spreadsheet data into the system described in this document at a future date.

From a design perspective, along with maintaining basic inventory information, we would like to organize information in a number of parallel hierarchies, for example location, power, control and cabling hierarchies.

From a construction/installation perspective we would like to easily add information to the system as items are received, tested and installed.

From an operational perspective we would like to include all information needed by technicians for all installed items they might have to deal with (test, modify, repair, loan out, etc). We also would like to maintain a spares inventory. Items in the system will include the usual crates, boards and power supplies as well as infrastructure items such as electrical panels, UPS's, and gas racks.

In the near term we need:

- Inventory information, such as item name, serial number, location, condition, repair status, loan status, documentation, comment, etc.
- Housing information, such as housing required or provided.

- Power information, such as power needed or provided.
- Control information, such as control interface needed.
- Cabling information, such as connectors needed or provided.
- History information, such as a log of changes made to an item.

In the longer term we would like to capture not only more extensive information in the categories above, but the relationships among them. For example the system should not let someone enter information indicating that a PLC module is installed in a VXS crate. Also, we can use the information in the database to generate module lists that front-end processes can use to verify all modules installed in a crate are actually responding. It is even technically feasible to generate EPICS database files and PLC programs from the component database, but we likely do not want to go to this extreme.

The graphical system should be web-based and able to display the information in the tables in a user-friendly form. This would typically be in tabular form for lists of similar information, or in document form to display full information about a particular item. It must be easy to add new items and modify and delete existing items, and it must maintain a history of such changes. Simple and advanced search capabilities are required. Note that it is difficult to envision all possible graphical needs so the system must be flexible enough to add new or modify existing graphical displays as needed.

A programming interface is needed to perform bulk loading, modification or unloading/backup of information in the database. This is typically not a problem with relational databases as they provide numerous ways to do this using multiple languages and system facilities.

The underlying database must be manageable using standard techniques and facilities, such as those provided by MySQL.

Finally, it would be useful to have a graphical representation of experimental areas that one could click on to get details of the equipment in that area, eventually drilling down to get full component information for a particular item. CLAS developed such a system as an adjunct to their inventory system, but it required a lot of hand coding.

## Current Status

I have performed preliminary evaluations of three systems: IRMIS3, CED and the CLAS systems.

The IRMIS project has been in development in the EPICS community since at least 2005, and unfortunately the latest version is still under development (previous versions are not maintained). Given our near term requirements IRMIS3 is out of the running. Note that we might consider migrating to IRMIS3 once it matures.

CED was developed by the JLab Accelerator Division. The schema design is mature, flexible and sophisticated, but graphics development is not well advanced, at least based on our requirements and our need for rapid deployment. CED is an excellent choice based on the underlying database implementation, but extensive graphics development would be needed to make it work for us, and

there just isn't time. One additional concern is that CED only runs under Oracle, but we likely could deploy CED on an upgraded Accelerator Division Oracle cluster so we would not have to perform Oracle system administration ourselves. Note that CED is a possible candidate to migrate to in the future.

The CLAS system started out as a freeware home inventory system that was substantially modified to meet CLAS needs by Sergey Boyarinov and Sergey Pozdnyakov. It was developed over a few year period and has been in use since 2008. It uses MySQL as the underlying database system. Not surprisingly it meets most of our near term requirements since it was developed with a similar problem in mind. The underlying schema is simple and not particularly flexible, but it is acceptable, particularly given our near term deployment requirement. The graphics are very good, implementing most of the functionality we need in the near term.

## Recommendations

I propose we adopt a slightly modified version of the CLAS system immediately and consider migrating to a different system if and when a better system appears and we have a compelling reason to switch.

The initial modifications I propose are similar to those CLAS made to the original home inventory system, so making them should not be difficult, especially with help from CLAS. The modifications mainly involve adding new fields to the base item table via ALTER TABLE commands (to maintain an upgrade history) and modifying the web templates to display them.

CLAS has expressed interest in adopting our modifications, so these modifications will likely be made to the CLAS system as well. It is clearly advantageous to both halls to use the same system and the same code base, so I plan to develop the modified system in the 12GEV repository and do everything in an experiment-independent manner. In this way CLAS can test new versions without breaking their existing system and upgrade if desired. CLAS can of course make modifications to the base system that we could evaluate and adopt as well.

## Near Term Plan of Action

I will first create a new section in the 12GEV repository named JInventory and check the CLAS system into it. Next I will clean up the code and consolidate all the CLAS-specific parts into a single file. At this point, after some necessary system changes on halldweb1, I will deploy our copy of the CLAS component database and we can start using it. Note that in this way other halls could easily deploy the system if desired.

Next I will create ALTER TABLE scripts that add new columns to the base Item table (see Appendix 1 for a list of existing columns, and Appendix 2 for a list of proposed additions). Then I will modify the PHP scripts and SMARTY template files to display the new columns.

Afterwards I will develop scripts as needed to allow for bulk loading of data from spreadsheets, files, etc.

Finally, concerning the longer term, I've discussed possible schema modifications with CLAS, including modifications they have been thinking about. These would require more extensive changes than just adding a column to a table and adding a field to a gui, so they need to be carefully considered and implemented. Such features could use the information in the new columns to display data in new and more useful ways (see Appendix 3 for a technical discussion concerning implementation of hierarchies in relational databases). These would be implemented so that all halls could adopt them.

## Appendix 1 - Existing Columns

ITM_ID int(11) NOT NULL auto_increment,
ITM_ShortName varchar(50) default NULL,
ITM_Description varchar(250) default NULL,
ITM_Picture varchar(50) default NULL,
ITM_Loc1Id int(11) default NULL,
ITM_Loc2Id int(11) default NULL,
ITM_Loc3Id int(11) default NULL,
ITM_Loc4Id int(11) default NULL,
ITM_Loc5Id int(11) default NULL,
ITM_LendToId int(11) default NULL,
ITM_LendDate date default NULL,
ITM_InsertDate datetime NOT NULL default '0000-00-00 00:00:00',
ITM_Quantity smallint(6) default NULL,
ITM_BrandId int(11) default NULL,
ITM_PartNum varchar(40) default NULL,
ITM_SerialNum varchar(20) default NULL,
ITM_PropertyTag varchar(20) default NULL,
ITM_State tinyint(4) default NULL,
ITM_Condition tinyint(4) default NULL,
ITM_ConditionDescr varchar(80) default NULL,
ITM_PurchaseDate date default NULL,
ITM_PurchaseLocation varchar(40) default NULL,
ITM_PurchasePrice varchar(15) default NULL,
ITM_CustodianId int(11) default NULL,
ITM_OnCallId int(11) default NULL,
ITM_ReplacerId int(11) default NULL,
ITM_OnSiteEvalId int(11) default NULL,
ITM_OnSiteRepairId int(11) default NULL,
ITM_OffSiteRepairId int(11) default NULL,
ITM_WarrantyInfo varchar(100) default NULL,
ITM_History text,

## Appendix 2 - Proposed New Columns

ITM_FormFactor varchar(64)
ITM_Function varchar(64)
ITM_PurchaseReq int(11)
ITM_Documentation varchar(256)
ITM_Comment text

ITM_Power_IFR varchar(64)
ITM_Power_IFP varchar(64)
ITM_PowerParent int(11)

ITM_Housing_IFR varchar(64)
ITM_Housing_IFP varchar(64)
ITM_HousingParent int(11)

ITM_Control_IFR varchar(64)
ITM_ControlParent int(11)

ITM_ConnectorInput varchar(132)
ITM_ConnectorOutput varchar(132)

# Appendix 3 – Hierarchy Implementation Plan

After a brief introduction to modeling hierarchies in relational databases I present tentative plans for upgrading the database schema to optimally handle hierarchical data.

How to represent hierarchical data is a well-known problem in relational database design. Early on it was thought that relational databases would never become popular since all existing databases were hierarchical and representing hierarchies was not natural in the relational model.

Four methods are popular today:

In the Adjacency List Model every entry in the base table has a parent pointer, thus you can reconstruct any piece of the hierarchy by following the parent pointers back to the root. Unfortunately this involves recursive queries and can be slow and cumbersome, especially for deep hierarchies.

In the Nested Set Model you maintain a triplet of pointers, one to the parent and two more to represent where the item resides in a set-theoretical mapping of the full hierarchy. This model has many advantages but is complicated to implement and can be slow to modify or update.

In the Materialized Path Model you maintain a complete representation of the path from the item to the root as one of the data elements of each item. This model too can be very slow to update as the whole table would need to be changed if the structure near the top of hierarch is modified. Also a lot of string or other manipulation is required to decode the representation of the hierarchy. Further, any integrity constraints have to be externally imposed.

Note that the CLAS system uses a modified version the Materialized Path Model, where instead of storing the path in a single complex field CLAS uses five simple fields, and thus is limited to a five-deep hierarchy.

In the Transitive Closure Model the base table maintains a parent pointer, as in the Adjacency List Model, but the limitations of the latter are avoided via use of an auxiliary table that stores relationships among the nodes. In particular, the auxiliary table contains entries describing each full path in the tree. The auxiliary table size can become prohibitively large in some cases. But entry and update operations become extremely simple and fast to perform.

The Transitive Closure Model is ideal for our case. We do not expect very large numbers of entries, we require fast retrieval and update, and the auxiliary table can be added after the fact based on parent data in the item table. All that is required is addition of a parent pointer for each hierarchy in the base table (see Appendix 2), then the auxiliary tables can be implemented when desired.

Finally, integrity for closure and other tables can be implemented via built-in database integrity constraints or periodic audits.