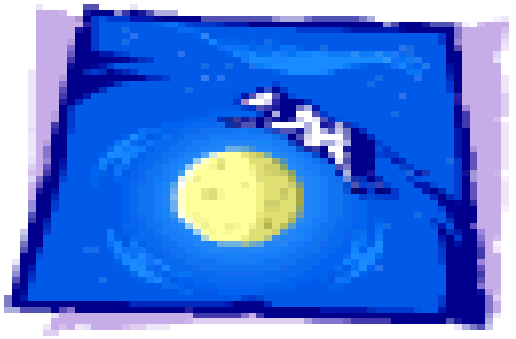
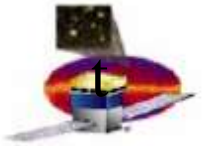
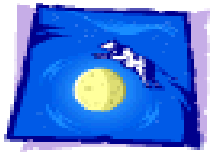


Configuration Tracking





Goals

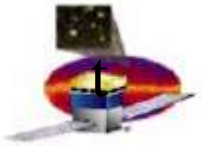
- Reconstruct the configuration used for a run
- Avoid creating and uploading redundant information
- Quickly implement something which satisfies above for upcoming calibration runs and smoothly evolves to handle test data runs and, ultimately, flight.

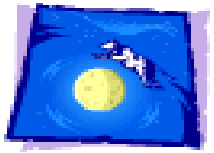
Reconstruct Config

- Input from data stream: identifiers (FMX logical_id) for files in use during the run.
- Work backwards from these through config file creation stages to recover intent.
- Only possible if MOOT participates along the way.

Conceptual Stages Involving MOOT

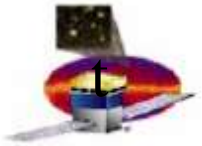
- Define and create a configuration. End product is collection of binaries.
- Determine destination, upload.
- Select among uploaded configurations; run.
- Analyze data

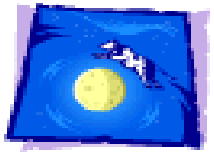




Implementation

- Identify information to work backwards to intent
- Design suitable database structure
- Implement services needed at each stage

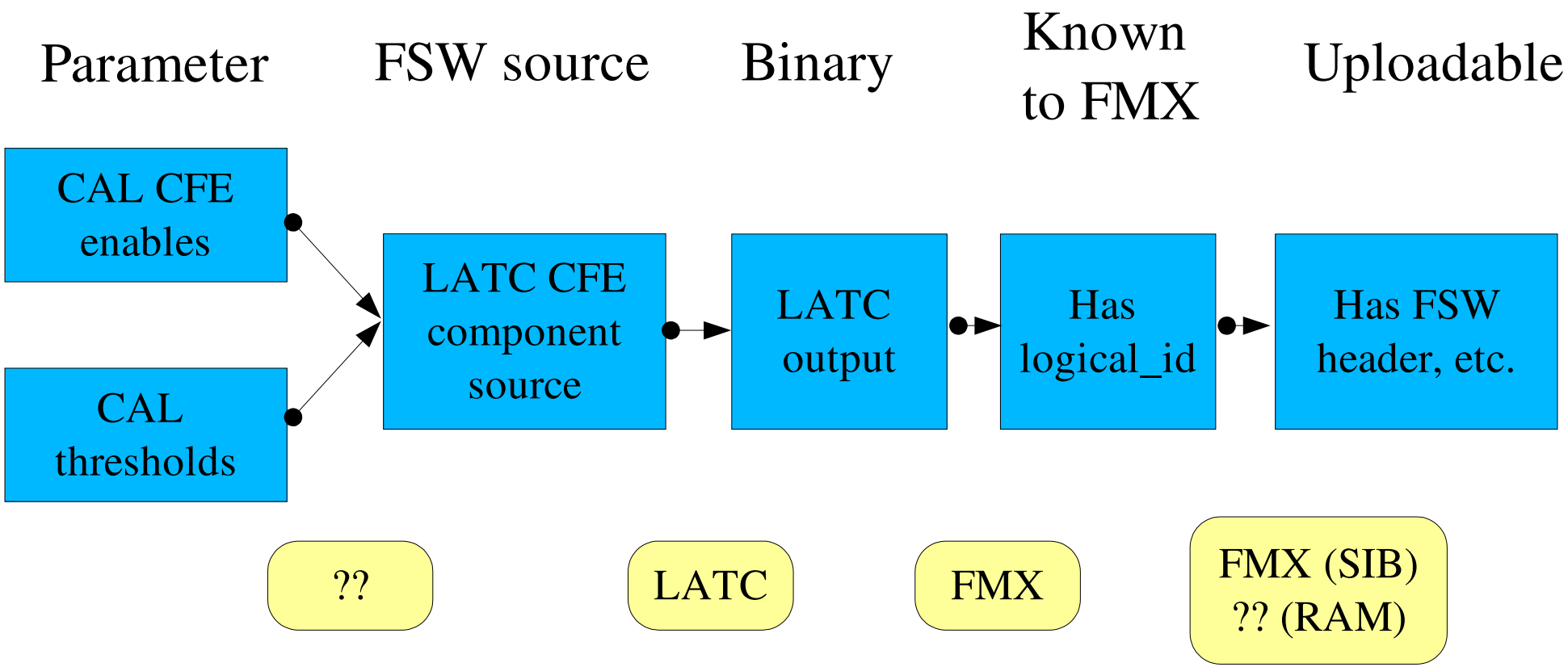




Config Contents

- Input to LATC
- Input to LCI for calibration runs
- Input to Filter, etc., for data runs
- For now, at least, MOOT will not keep track of thermal parameters, power-on state, etc. These can be added later if needed.
- MOOT will not keep track of code modules.

Config File Pipeline (MOOT view)



Only first stage may be many-to-one; all others are one-to-one.
Imagine parallel pipeline for each LATC component

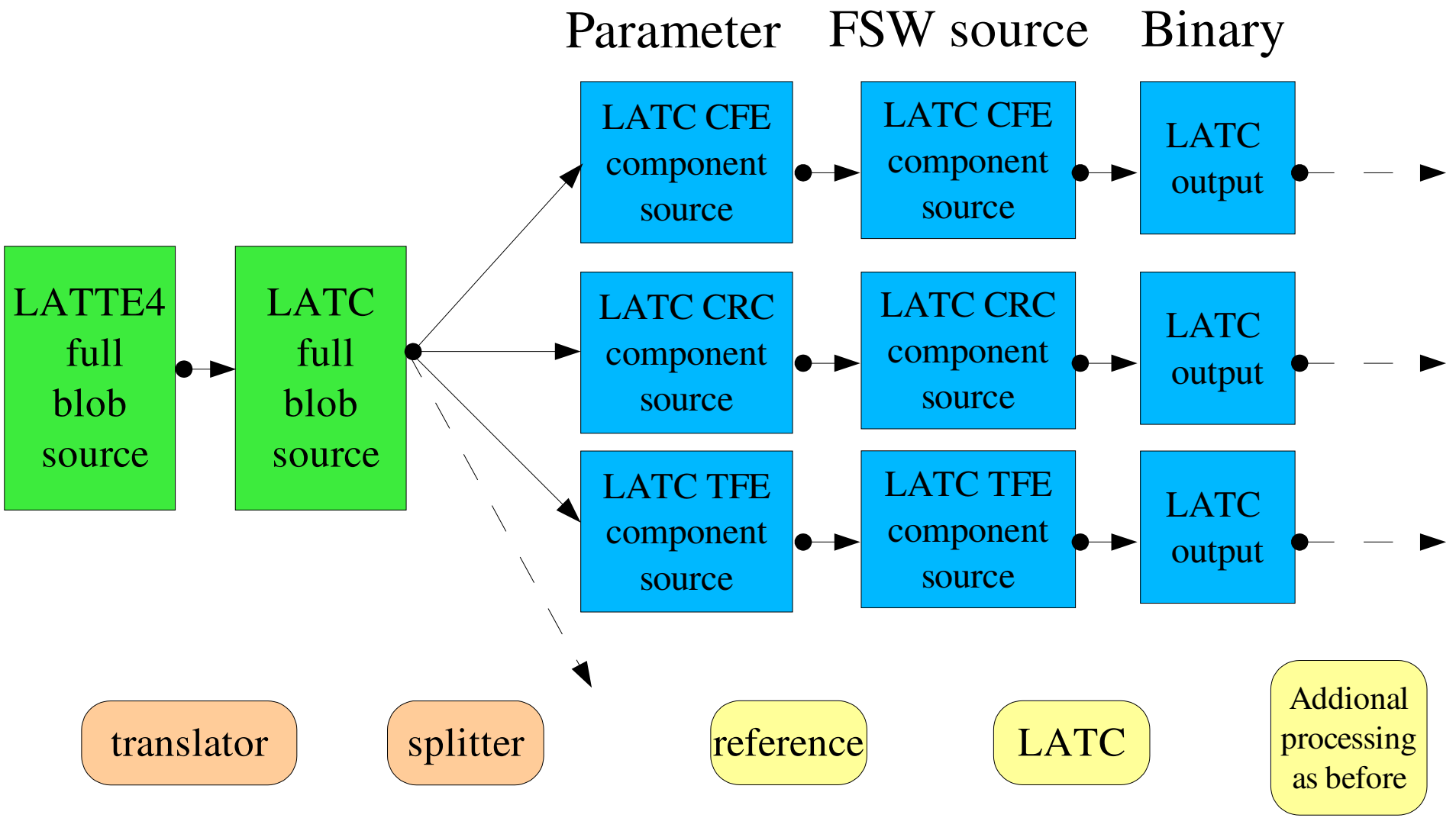
Plan for Calibration Runs

In addition to reducing definition of configuration to a minimum, will also simplify by insisting that there be only one Parameter file per FSW source, and in fact that they be identical.

This eliminates one of the ?? on the previous slide, and will make it easier to translate LATTE4 register descriptions to a form MOOT can handle.

Have lost some descriptive power, probably not needed here.

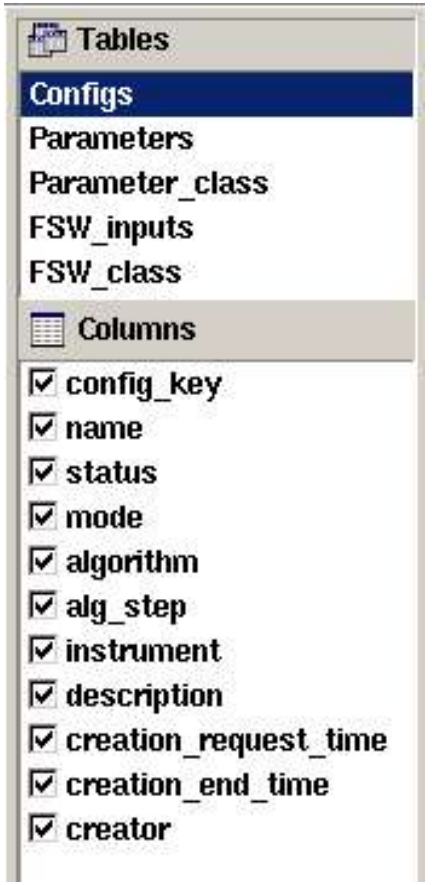
Pipeline for Calib Runs



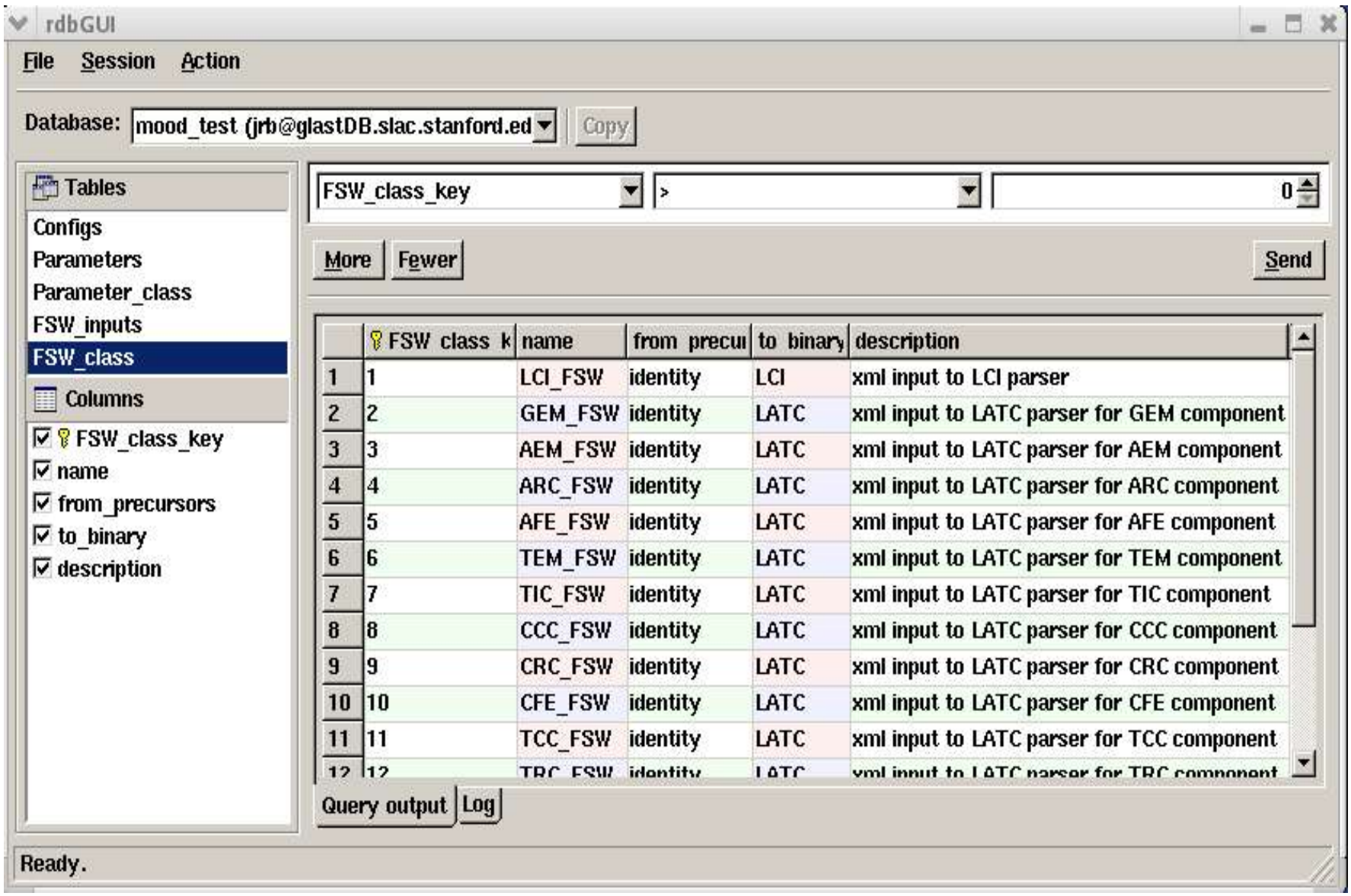
Database design

- Primary tables to represent Parameters, FSW inputs, and Configs (History table is gone).
- Secondary tables to describe classes of parameter files and classes of FSW inputs.
- Secondary tables to relate Parameter file instances to FSW input instances; FSW input instances to Configs.
- Design is done; still subject to minor tweaks

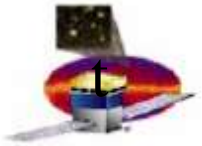
Primary Tables



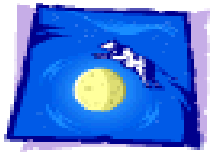
FSW File Classes



	FSW_class_k	name	from_precu	to_binary	description
1	1	LCI_FSW	identity	LCI	xml input to LCI parser
2	2	GEM_FSW	identity	LATC	xml input to LATC parser for GEM component
3	3	AEM_FSW	identity	LATC	xml input to LATC parser for AEM component
4	4	ARC_FSW	identity	LATC	xml input to LATC parser for ARC component
5	5	AFE_FSW	identity	LATC	xml input to LATC parser for AFE component
6	6	TEM_FSW	identity	LATC	xml input to LATC parser for TEM component
7	7	TIC_FSW	identity	LATC	xml input to LATC parser for TIC component
8	8	CCC_FSW	identity	LATC	xml input to LATC parser for CCC component
9	9	CRC_FSW	identity	LATC	xml input to LATC parser for CRC component
10	10	CFE_FSW	identity	LATC	xml input to LATC parser for CFE component
11	11	TCC_FSW	identity	LATC	xml input to LATC parser for TCC component
12	12	TRC_FSW	identity	LATC	xml input to LATC parser for TRC component



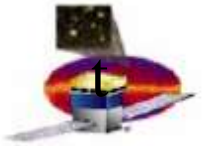
Conceptual Stages (again)

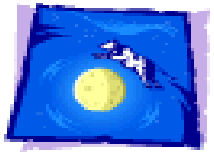


For details of implementation for services MOOT will provide during the first three stages – configuration creation, transport, and use – see http://www.slac.stanford.edu/exp/glast/ground/notes/moot/ConfigSteps.shtml
Or see the excerpt in the next slide, probably more than you wanted to know.

MOOT has a lot to do with building a configuration but is only a very minor player in transport and use.

Support for the fourth stage – data analysis – flows from the database design. It contains the information needed to relate the logical ids in the data stream to parameter files.





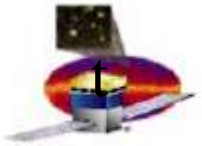
For each LATC file created {
  // If it's a duplicate just retrieve relevant information, otherwise
  // have to make a bunch of entries in MOOD and invoke FSW services

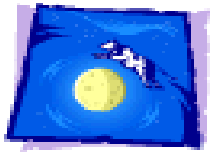
```
// when the print gets small and ugly like this you can probably skip it
// calculate a checksum this will help when looking for duplicates

search Parameters table for match of Parameter_class value and checksum value;
If (find any)  {
  bool match=false;
  for each found  entry {
    retrieve file corresponding to the db entry, do diff against our file
    if match {
      handleMatch(...  );  // gets and saves information about existing entry
      match=true;
      break;
    }
  }
}
if (!match) {
   makeNew(.. ); // makes a dbs entries, creates binary, does fmx add
}
Make entry in Config_FSW table so we know which files belong to this config;
```

  // At this point, have key for this file in MOOD Parameters table,
  // MOOD FSW_input table, and have FMX logical_id
}

Status

- Non-MOOT preprocessing (Byron's translator, Jim's splitter) is well in hand or done.
- Database design is done; will continue to tweak.
- Detailed design and implementation of MOOT services underway.
- Hope to have enough code written in a week or two to go through some of the motions.





Input Needed

- Confirmation that the path outlined is adequate to satisfy requirements and in tune with reality. If there is a high-level flaw, I need to know it now. Changes in details are not a problem.
- Communication: need to be kept in the loop about changes made to anything MOOT touches.
- Analog of *fmx upload*, *fmx commit* for uploads to RAM.
- Guidance on preferred environment for building code.