

---

# Cadence-Perforce Integration

Shiv Sikand

Silicon Graphics Inc.

Visual Systems Group - Advanced Graphics

sikand@engr.sgi.com

---

## **Data Management Issues**

- Proprietary CAD database
- New architecture in version 4.4 allows Unix based control of files
- Multiple, co-managed files
- Available shrink wrapped solution - Team Design Manager
- gdm available in 4.4.2
- Custom solutions also available from Cadence Spectrum Services

---

## **Why not TDM ?**

- Base TDM is similar cost to a Perforce license
- Addittional cost for TDM Design Project Admin
- Many open PCR's, high support load
- Not well suited for use outside Cadence environment
- Low performance/high complexity

---

## **Perforce suitability**

- Common SCM for source design files and physical design data
- Change/Submit model maps closely to Skill trigger model
- Atomic checkins handle co-managed files naturally
- Core interface code is only ~500 lines of Skill, elegant solution
- Change accounting eliminates need for complex tracking code to handle dependencies
- High performance/low complexity

---

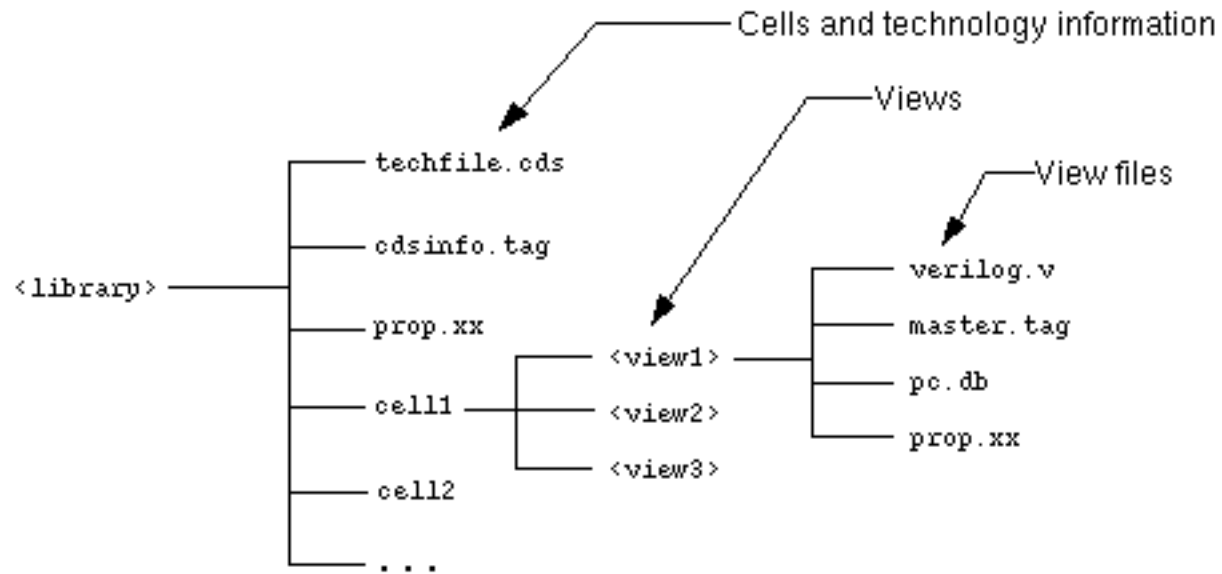
## Model differences

- TDM uses linking and only brings the file into the workspace on edit rather than read
- With Perforce, we use specific clients for system and shared libraries to prevent multiple copies in user workareas
- Requires some planning so that interface knows how to set up clients without manual intervention.

---

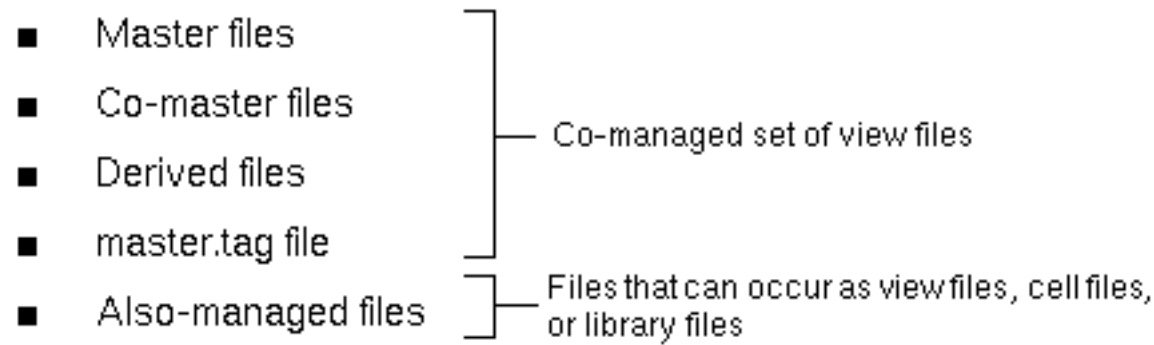
## Cadence Library Structure

A library consists of cells, views, and technology information.



---

## Cadence Library Structure



- Master file contains source data that cannot be re-created without user input. One master file per view, contained in the master.tag file
- Co-master file contains source data that is useless without the master file.
- Derived files can be re-created from master files, or master and co-master files, but only the creating tool can distinguish between them and co-masters.

---

## Integration techniques

- Skill interface to database files identifies names, paths and file type
- ddLibFileType for library files
- ddViewFileType for view files
- ddCellFileType for cellview files
- For cellview files, checkins/checkouts are keyed on the master file, typically \*.cdb
- The autocheckin/checkout behaviour is controlled by UNIX variables

eg CDS\_PROMPT\_CKIN=views

CDS\_PROMPT\_CKOUT=none

CDS\_AUTO\_CKIN=files

CDS\_AUTO\_CKOUT=all

---

## Available triggers

PI function	Pre-Hook name	Post-Hook name
ddUpdateLibList()	“PreUpdateLibList”	“PostUpdateLibList”
ddGetStartup()	“PreGetStartup”	“PostGetStartup”
ddDeleteObj() ddDeleteLocal()	“PreDeleteObj”	“PostDeleteObj”
ddCreateLib()	“PreCreateLib”	“PostCreateLib”
ddGetObj()	“PreCreateObj”	“PostCreateObj” “FirstAccessLib” *
ddCheckin()	“PreCheckin”	“PostCheckin”
ddCheckout()	“PreCheckout”	“PostCheckout”

---

---

## System issues

- 'cdmcheckin' binary must be replaced with your own version
- In SGI solution, this is a binary which always returns exit 0 since entire interface is handled in Skill
- Allows external binary to handle data directly. Flags passed to cdmcheckin are not documented but can easily be instrumented
- Use new 4.4 higher performance IPC interface
  - eg ipcBeginProcess() vs hiBeginProcess()
- Autocheckin overrides settings for library file checkins
- Checkin and checkout routines determine success of action based on file permissions

---

## System Issues

- Watch out for NFS filenames -- write mapping routine that resolves a path to a common name on all machines to avoid p4 name conflicts
- Need to build your own library manager since existing library manager does not call triggers correctly (fix due in 4.4.3) or allow for Skill based user customisation ( eg Show Checkouts)
- Make sure that you set P4CLIENT before performing any p4 commands
- Use `p4 -d <directory>` as needed since `$PWD` is not correct under IPC

eg `changeWorkingDir()` does not update `$PWD`, it is always returned as the directory from where you started the tool.

---

## Client setup

- Library properties determine client setup
- Two level model
- System libraries - shared data area, multiple users
- User libraries - multiple clients for multiple users, clients created using p4 client -t option
- p4 protect used to setup permissions for all system libraries

---

## Database objects

- Database identifiers provide file data

type name readpath writePath lastModify

owner ownerAccess group groupAccess publicAccess

isReadable isWritable prop lib cell

view

---

## Library level triggers

`ddRegTrigger("PreCreateLib" 'SGI_P4PreCreateLib)`

Args: library name, full path to dm.tag, ddid

- Create a client for the library based on user preference

`ddRegTrigger("PostCreateLib" 'SGI_P4PostCreateLib)`

Args: libname, libpath

- Register the DM system using `ddSetLibDMSys`
- Add/submit dm.tag
- Setup properties to determine client behaviour
- Autocheckin/checkout activates as long as the contents of the dm.tag file is not "none".
- If 'cdmcheckin' is not replaced, you will get "Unknown DM system xx" messages.

---

## Object creation triggers

ddRegTrigger("PostCreateObj" 'SGI\_P4PostCreateObj)

Args: ddid, cell path, ddtype, childid

- Check ddType to determine behaviour
- Setup p4 type - text or binary based on file name
- Perform add/submit action for lib files and cell files
- For view files, perform add only
- Schedule add for master.tag file when a .cdb file is detected

---

## Object Deletion Triggers

ddRegTrigger("PreDeleteObj" 'SGI\_P4DeleteObj)

Args: ddid

- Use pre-trigger!
- For ddLibType and ddCellType do nothing (directories)
- For view files, schedule delete or revert
- Key submit of deleted view files on .cdb file for atomic change
- Check contents of master.tag file before deleting since object master may have been changed by tool
- Reject p4 deletion of .cd% backup objects

---

## Checkout trigger

ddRegTrigger("PreCheckout" 'SGI\_P4PreCheckout)

Args: ddid list, options

- Detect .cdb file and get atomic list of files associated with last checkin and mark for edit
- For all other checkout requests, mark appropriate file for edit
- Property bags checkouts are handled automatically by requesting a checkout for .cdb object

---

## Checkin trigger

ddRegTrigger("PostCheckin" 'SGI\_P4PostCheckin)

Args: ddid list, description, options

- Check for "c" option flag which signifies a forced cancel of last checkout
- Key derived and co-master files on .cdb file
- execute a p4 diff -sa to check the file is different before submitting
- Check status of master.tag and its contents as well as status of prop.xx and pc.db files
- Add them to changelist if they are open for edit or add to ensure atomic transaction

---

## Atomic transactions on .cdb objects

- Use p4 filelog to get file history
- Parse output to get change number for last edit or add operation
- Use p4 describe -s and parse output to get affected file list

```
status = SGI_P4IPCBufferProcess(strcat("p4 filelog " ddid~>readPath))
```

```
when( status == 0
```

```
    changes = parseString(SGI_IPCBuffer "\n")
```

```
    ;; get the files associated with the last edit
```

```
    changetext = car(setof(x changes rexMatchp("change [0-9]+ edit" x)))
```

```
    when( changetext
```

```
        ;; get the change number
```

```
        changenum = nthelem(4 parseString(changetext))
```

```
)
```

---

```
;; or failing that, the last add
unless(changenum
changetext = car(setof(x changes rexMatchp("change [0-9]+ add" x)))
when( changetext
  ;; get the change number
  changenum = nthelem(4 parseString(changetext))
)
) ; ** unless changenum **

if( changenum then
;; parse the change description to get the affected
;; files
status = SGI_P4IPCBufferProcess(strcat("p4 describe -s " changenum))
```

---

```
when( status == 0
    desc = parseString(SGI_IPCBuffer "\n")
    foreach( dline desc
when( rexMatchp("^\\.\.\.\.\" dline)
    file = nthelem(2 parseString(dline))
    rexCompile("#[0-9]+")
    rfile = rexReplace(file "" 1)
    rfiles = strcat(rfiles " " rfile)
); ** when rexMatchp **
    ); ** foreach dline **
```

---

## Submitting files

- Cannot rely on default changelist to contain the right state except for library creation
- Use p4 opened to get the depot names and add in the co-managed files if they are in an open or edit state for .cdb files
- Build the changelist using p4 submit -i

---

## **Management functions**

- Incorporate show checkouts, cancel checkouts, synchronize and other version information using CIW menus or on custom library browser
- Release labels and configurations are done from Unix using native commands rather than building Skill GUI interface