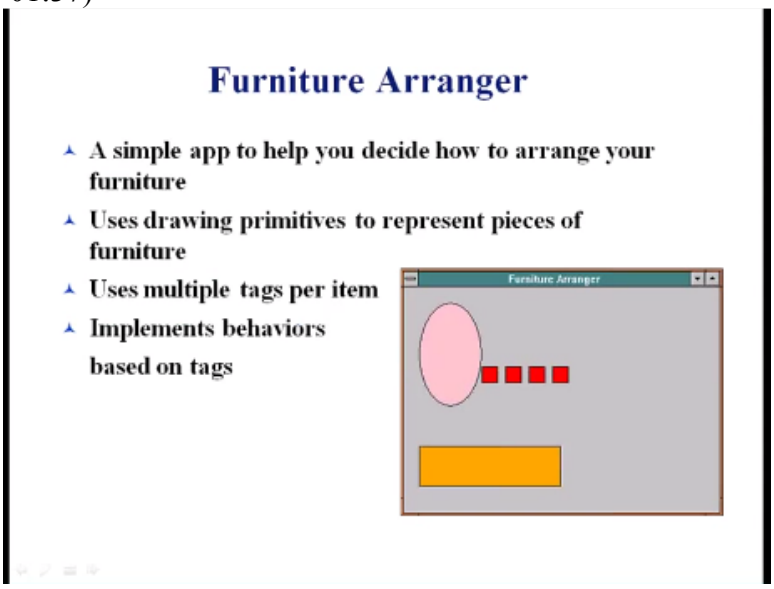
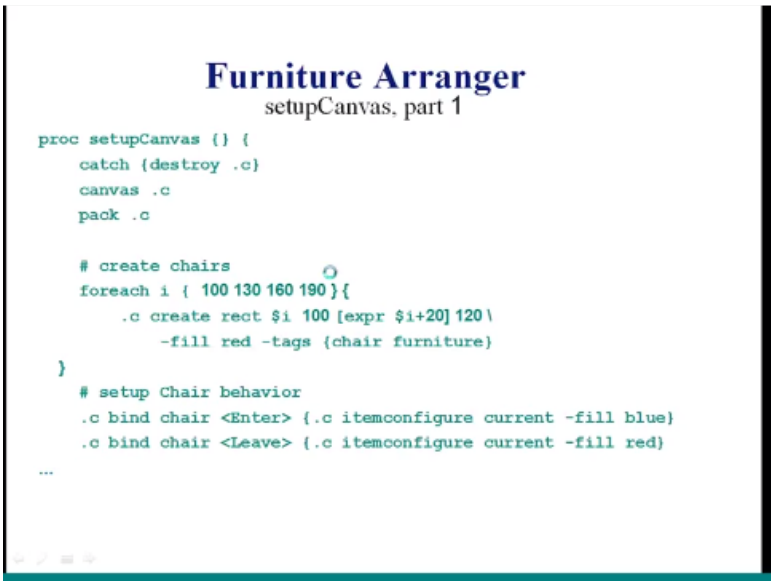
Hi everyone ,again we are continuing the LPS class and today ,we are continuing with the Tk session ,so in the last , lecture we saw some of the additional commands of Tk , when we talked about the channels, how to create the canvas, what are the attributes or to what are the bindings of the tags and all those things today, we are going to look at one example of using canvas and this is one application, that we are going to build.
And then I am going to take you ,walk it through the application and see like I mean can make use of some of these concepts ,that we learnt in our class, so again the canvas we did is, it manages 2 D collection of graphical objects lines ,circles , images, we actually went and like put some ,smiley face with a circle and then two small circles, with black color in it and then we also drew like some arcs to represent the mouth and more .
So we actually now have a fairly good understanding, as to how to draw this, so I am going to go through this particular example.
(Refer Slide Time: 01:37)




So the application, that we will be building is this, furniture arranger application, it is a simple app to help you to decide, how to arrange your furniture and we also use some drawing primitives ,to represent pieces of furniture, so here there is an oval ,some squares and rectangles, so these represent the furniture top down view , same and then we will use multiple tags for item and then we will implement the behaviors, based on the forms .So let us see what how we can do it.
(Refer Slide Time: 02:12)

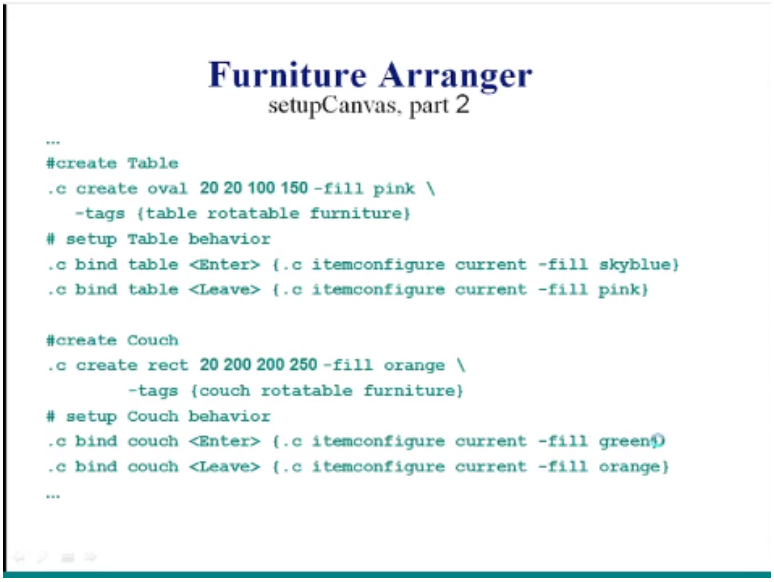



So we define some procedure ,so here there is a procedure setup canvas, this is essentially like I mean ,first erase everything ,all the windows attributes and then basically like we will build the canvas, so create canvas command, create the canvas and the widget is . c and then we take pack man and then we create those chairs , so those are the small squares here ,so we create them.

So for each of these numbers essentially ,that I am going to create a small rectangle and essentially like I mean, that is with the actual location basically will represent , so and then we fill the thing ,with the red and then we tags as chair and furniture, so as in the furniture big picture, this is indicates the chair and then now we will set up the chair behavior, for that we use the bind commands and then the bind command is essentially ,capture the event ,enter and then we say basically current .

So basically ,we take the chair to the current location and then fill it and then if the leave command is given, then we basically like moving ,we can then fill it again ,so that it can be moved .

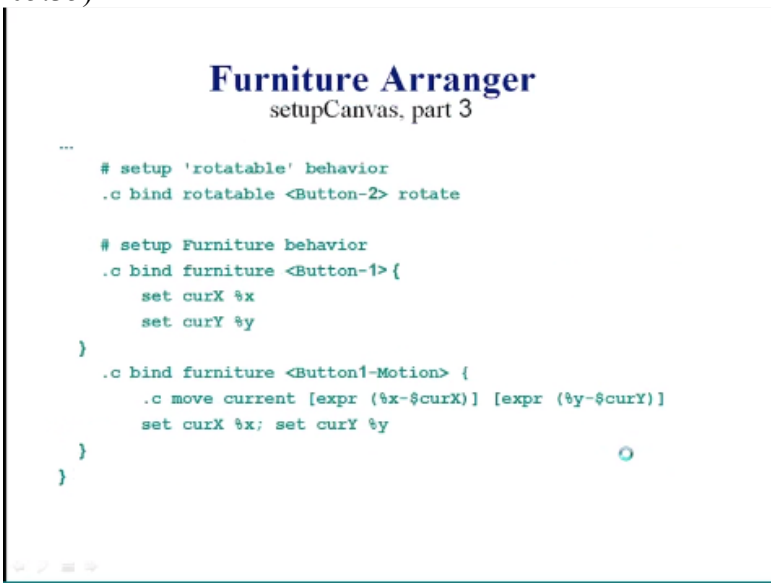
(Refer Slide Time: 04:19)



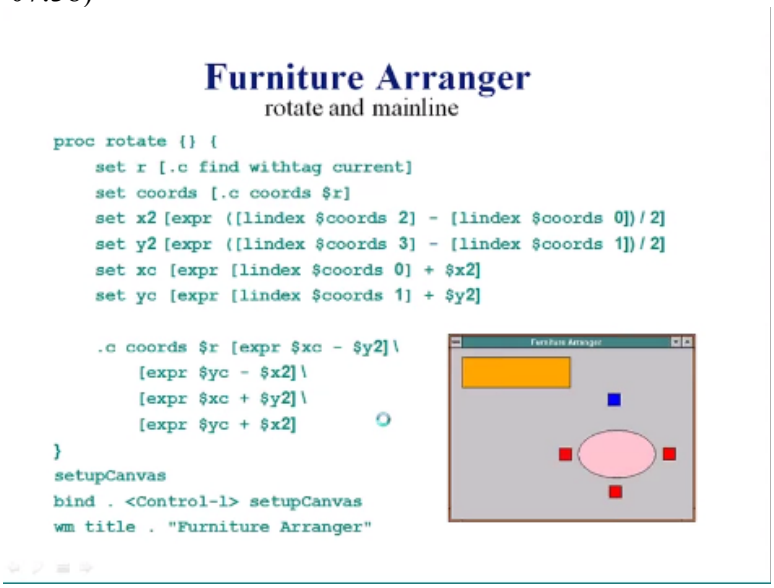

Now the second one is, we create the table ,that is ,those the oval-shaped figured here basically, so for creating the table again ,we create the oval and now we fill it with pink and then ,we create some tags for it ,which ,is table rotatable and furniture and then again, we same similar kind of binding, that we create which is the table behavior is basically based on these events.

So it turns into sky blue and then back to pink and then the couch is actually , represented as a rectangle in the picture ,this is this one, so a chair, the table and the couch , so the couch is actually filled with orange and then we have a couch ,is also repeatable and then it is a constant and then ,we have same way ,we define the bindings and then here the binding is filled with green ,instead of orange.

(Refer Slide Time: 05:35)

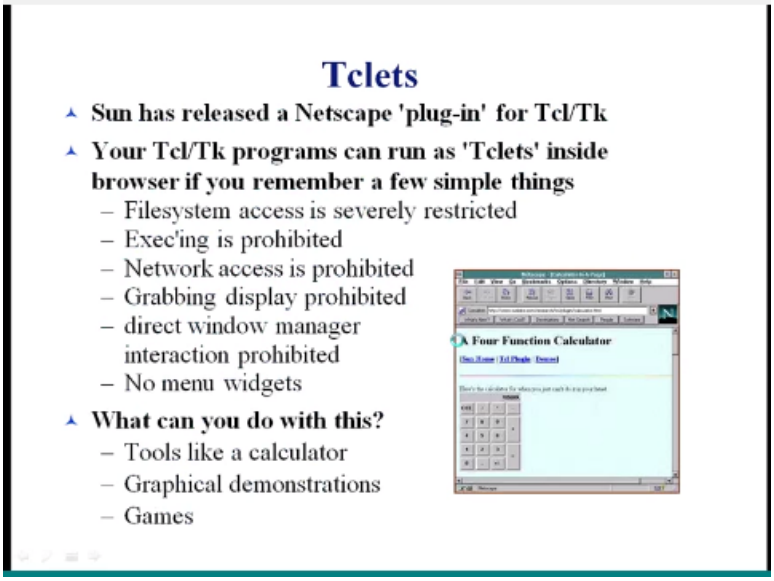


Now ,we need to also , since we are generating tags which are rotatable, so we need to define the tags also, so here we know notice that actually go wrong , so here we denote fault in some bindings here , so the rotatable is essentially making , we bind it to button 2 and then that causes it to rotate, so this is one of the most click buttons, and then we also need to set up the furniture behavior , because that is the highest hierarchy , highest level of hierarchy in this tags basically , this is all furniture, this is furniture and then look at chair -chair is also furniture.
So the way that ,we define the furniture behavior is the button 1, there be what is the current x and current y okay ,so and then we also generate the motion ,which is like going to click on it and then you can grab it ,to move the particular furniture ,that move is represented as basically ,we say that it is move from current ,to or basically like I mean the current position is moved to whatever ,we captured the previous ones, the % x and %y from there to the current x and current y .
So we just add that and then that is ,what your new position is and then that is set as a new current x and current y, so once we define this, all these procedures, so we are ready to actually move and then .
(Refer Slide Time: 07:38)



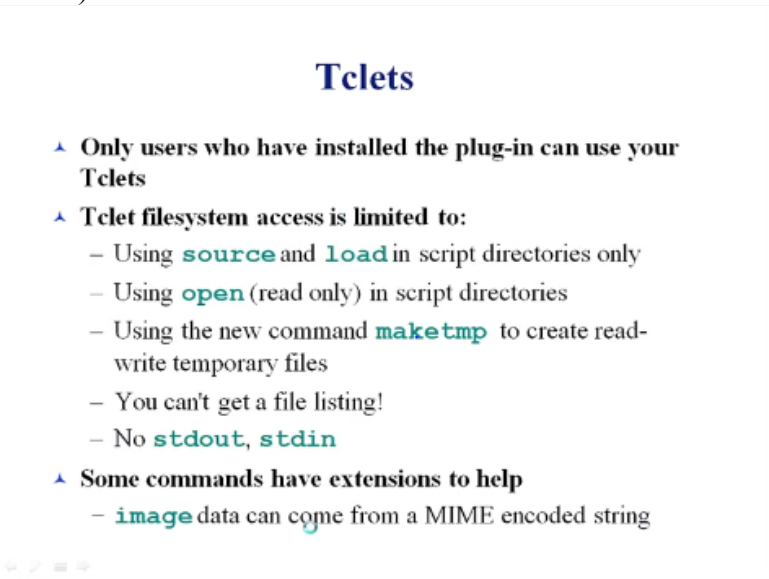

Now we also want to do , some rotate ,so the procedure rotate is essentially ,we find what is the element that from that window, like what is the selection and then, we also like set the coordinates ,to that particular coordinates and then we basically move it ,or essentially like find the new coordinates based on these formulas ,we subtract the original 1 by 2, so now that becomes the new coordinate essentially ,we do some more manipulation and that is the rotate function basically.

What that means is ,that you can grab something and then rotate it and which other way that you want ,so that is what this procedure will do ,now that we define all the , we can now set up a canvas, basically using set up canvas, which is already defined as the procedure here ,which is what creates the canvas and then we bind the top-level window to control ,now l and then that sets up the canvas and then we basically put the title as furniture arranger which is right here.
So it is a fairly simple program, I think you should be able to just, go through it and understand this.
(Refer Slide Time: 09:48)




So now ,let us look at some more topics , essentially one thing that I want to introduce you to is this Tclets, Tclets are just plug-ins for your browser ,so that you can embed a TK program inside the browser, in the browser XML and that actually ,now and work just like any other TK program ,so there are some Netscape plug-ins which I do not know, I mean for what you know you can look at but mainly like, I mean the programs run as Tclets inside the browser .
And it needs ,you need to remember two things ,one is the file system access is severely restricted, so you do not want to try to access some files, there you do not have permission for and then exec ,using exec is prohibited ,so you do not want ,to use that and the network access is also prohibited.
So you cannot go and find things in the network and grabbing display is also prohibited, because you do not want to grab display from during the execution of the program ,so that it will never come back ,and then the direct window manager interaction is also prohibited, so you do not want to interact directly ,with the window manager for anything and then there are no menu widget.So as long as ,you can keep track of this, you can write your own Tcl or Tclets, which is

embedded TK programs and what do you ,what can you do with Tclets ,it is basically like you can design tools like calculator, on the file, we can do like graphic demonstration from games, design games in the browser in the nets .
(Refer Slide Time: 11:46)




So that is about Tclets and mainly like I mean for using it, to place the user , needs to have the install that plug-in Tclets file system ,access is limited to using source and load script directories, so those directories are open for you for the sources, or load those scripts essentially and then you have to use ,open as a read only in the script directory, so you can read from the script directory, not write to it .
And then using in the new command ,make temp to create redirect temporary directory ,so that is the bottom line basically like I mean ,so you can use only like these and then you cannot get a file listing and there is no standard in , some commands have extensions to help, for example the image data and come from a MIME encoded string .
(Refer Slide Time: 12:57)

## Tclets

- Only users who have installed the plug-in can use your Tclets
- Tclet filesystem access is limited to:

```
if {[info exists embed_args] == 0}{
    wm title . "Furniture Arranger"
}
```

- `embed_args` is an array containing parameters from HTML - you can use them as command line args for your Tclet


(Refer Slide Time: 12:58)

## Making 'Furniture Mover' a Tclet

- Only users who have installed the plug-in can use your Tclets
- One change to source:

```
if {[info exists embed_args] == 0}{
    wm title . "Furniture Arranger"
}
```

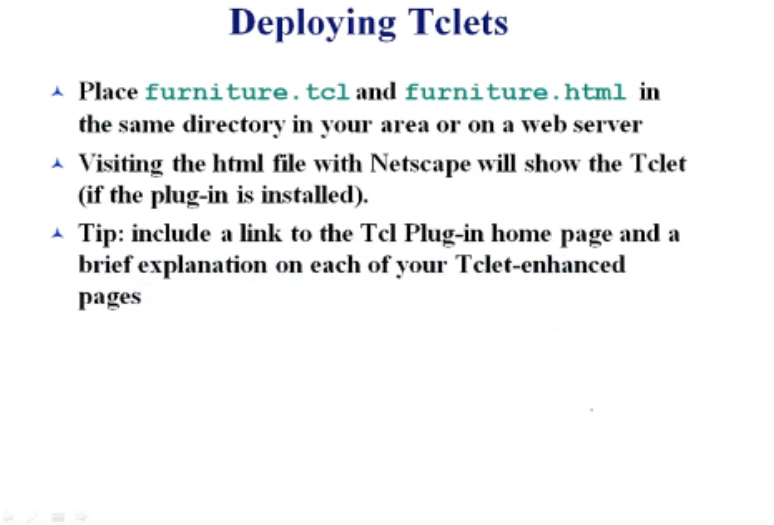
- `embed_args` is an array containing parameters from HTML - you can use them as command line args for your Tclet


So if you have to take the furniture mover and convert that convert that into a Tclets basically things ,that to do one is first of all the furniture mover application itself ,makes a perfect Tclets, anyone in the world could rearrange the furniture ,there is only one change to the source basically, this is the one, because if info exists embed _ args =0, then do the window manager ,the embed _ args is an array , containing the parameters from HTML ,you can use them as a command line arguments for the Tclets .

(Refer Slide Time: 13:42)

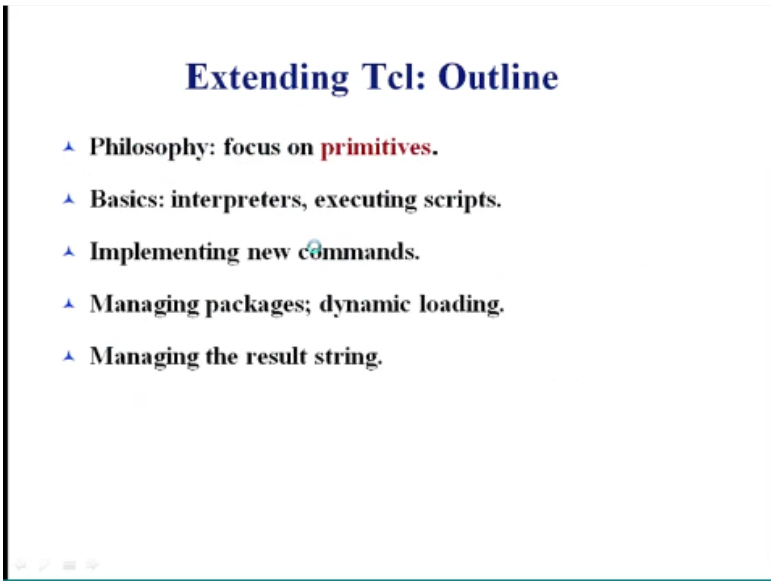

So let us create a HTML page, for running this furniture mover ,so here we call the title and then the body is essentially, let there be embed for the furniture .tcl and then ,we specify the width and height and then we specify what it does basically ,that is the import .

(Refer Slide Time: 14:12)




Now ,we need to place the furniture .tcl and the furniture .html in the same directory, or in your area or in the web server and then using the html file with even like any browser , will show the Tclets essentially ,if the plug-in is installed , so one thing that you can do is you can include the link to the tcl plug-in page, plug-in home page and a brief explanation of each of the Tclets enhanced page.

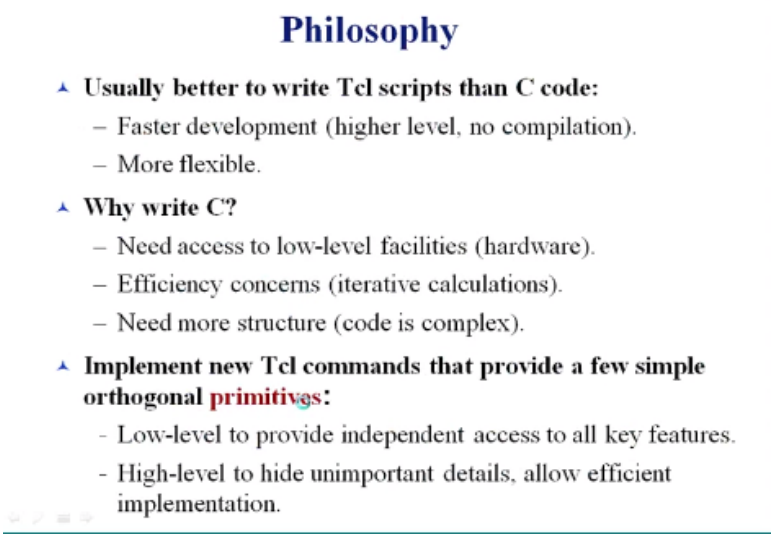
(Refer Slide Time: 14:52)




So that is pretty much on the Tclets, now we will go into the next topic ,which is essentially how to extend Tcl, so the basic philosophy behind that is focus on primitives ,some of the basics ,we already talked about the interpreters and executing scripts, we implementing new commands instead of the one and then the managing packages ,how do you move with dynamic loading and managing results string, which is how do you , pass the results.

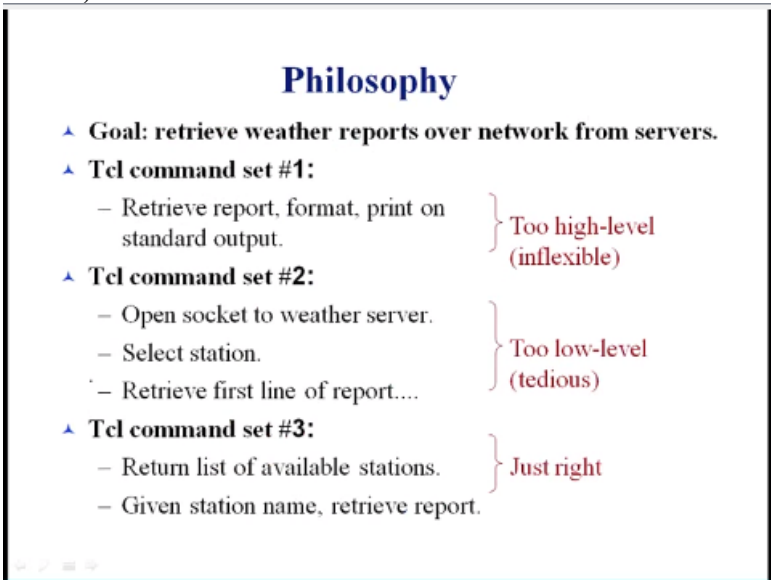
And then some of the usual library procedures like parsing, variables, list and the hash tables.

(Refer Slide Time: 15:44)

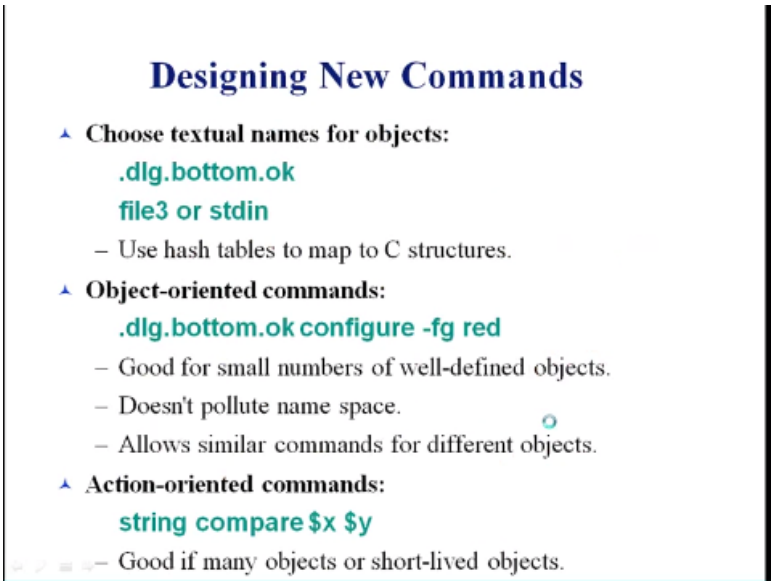



So in general the philosophy, basically it is better to write Tcl scripts than C code ,mainly because it is a faster development , also it is higher level and no compilation ,.so it is more flexible ,this section is ,what we saw when we introduced Tcl in the very early stages ,why write in C ,so if we need to access the hardware facilities or any kind of facilities provided by the

operating system then ,we need to use C and then if the program itself is big , then this way there is a lot more efficiency to gain ,then it is better to use C .
And the other thing is, when the code becomes complex ,the structure becomes much more important, so the way that definitely like I mean, we need to have some structure ,how to write with a subroutine ,again in that case C will be a better language , so implementing the new Tcl commands, that provide few orthogonal primitives ,there are some Tcl commands , that can actually give some extra one .
So the low level to provide independent access to all, the key features, the primitives and then it also high level of high enough level to hide the unimportant details and that allows an efficient implementation.
(Refer Slide Time: 17:27)




So here ,we are comparing like three different Tcls for weather reports ,so the goal is to retrieve the weather reports over a network from various service ,so the first Tcl command is to use the retrieve report format and print on the standard output, this is too high level and so it is inflexible ,now we can write another command which opens a socket to the weather server, select the station and retrieve the first line of the report ,this is too low level.
So again like I mean ,we may not use this command, even though this is provided in Tcl, finally the command set 3 basically the algorithm that is followed is to return the list of all the available stations ,given station name, retrieve report, so this kind of thing is just right.
(Refer Slide Time: 18:36)




So how do, we design new commands ,so a couple of useful tips basically choose textual names for objects ,so dialogue . bottom .ok and file 3 or stdin and then use the hash tables to map to C structure system, object-oriented commands essentially they are basically developed on bottom .ok and then use configure and then say foreground is red , so these kind of object-oriented commands are good for small number of well-defined objects.

And does not pollute the namespace and allows similar commands for different objects and then we also have action-oriented commands like string compare x and y, this one is useful if many objects are short lived for many objects or short lived objects this command.

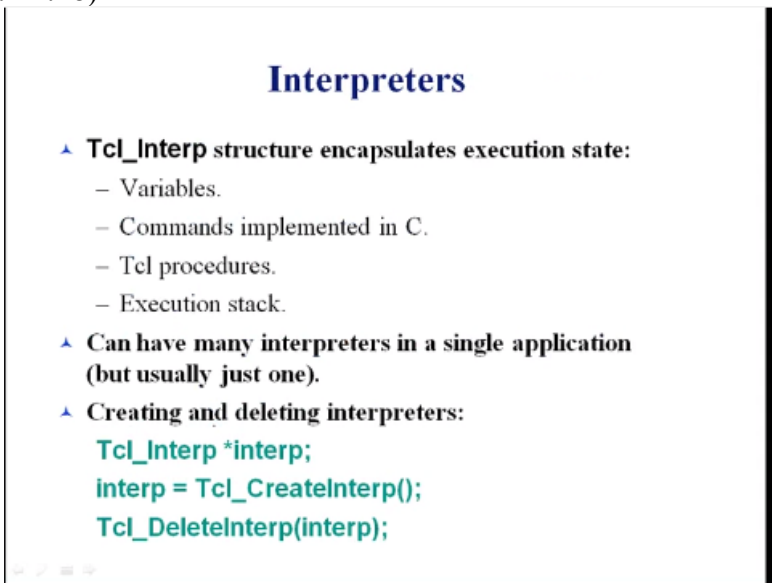
(Refer Slide Time: 20:10)


Now how do you format the command results ,couple of suggestion essentially like make them easy to parse with Tcl scripts, so here like temp 53 hi 68 low 37 precip .02 sky part , so here it is

very hard, if you are using these numbers ,so you got to convert them over to symbolic wherever possible ,so we do not use 53 68 37 .02 7 instead ,we need to replace it, by item low 10 things like that .

And then we can also use the package prefixes in command names and global variables to distinguish what where exactly it is coming from, for example weather stations denote the packaging information, for this particular command and weather report is another command and then midi play is another one, so these kind of naming convention can also allow packages to coexist without any name clashes .

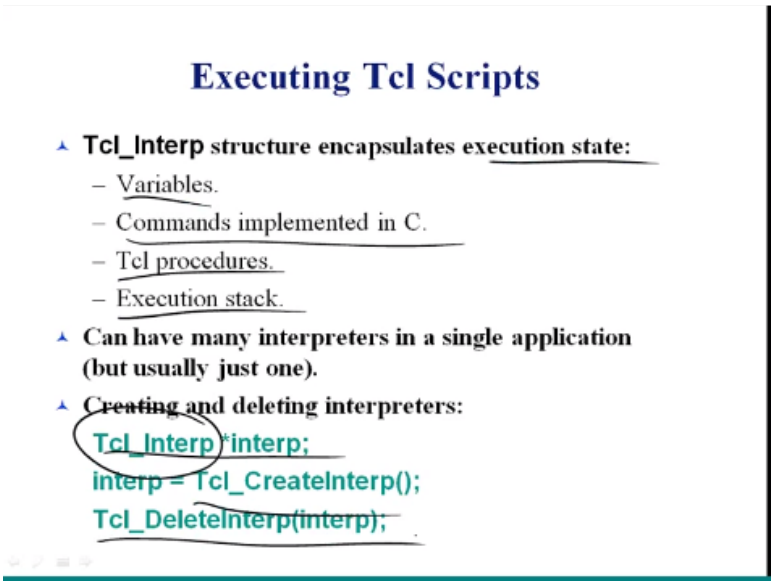
(Refer Slide Time: 21:48)




So now let us talk about the interpreters, again the Tcl Interp structure encapsulates the execution state after the variables, the commands implemented in C, the Tcl procedures and then the execution stack ,we can have many interpreters in a single application ,but just only one is active that and creating and deleting interpreters this in Tcl .

We can do like the * interrupt which creates an interpreter with the Tcl Interp command and then we can assign the interp to the Tcl create interrupt this is another way to create one and then we can also delete it like this doing Tcl delete command.

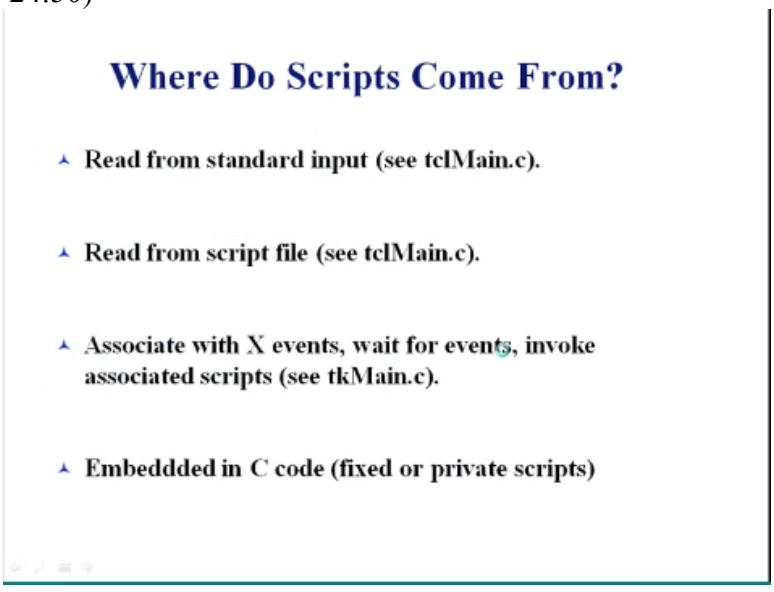
(Refer Slide Time: 22:51)




So now how do we execute the Tcl scripts, essentially, so here we have this variable declared and then we can use Tcl you have inter, with set a = 1, this is one which is Tcl script ,another one is essentially like , we can move put it into a file or init.tcl and then we can sat Tcl eval file interp and then file name and then finally we can also specify in a different way, which is using the var eval command and then here ,we give like the inter set a and 1 as 4 different objects, the code itself contains whether there is a success or failure and it comes to Tcl okay .

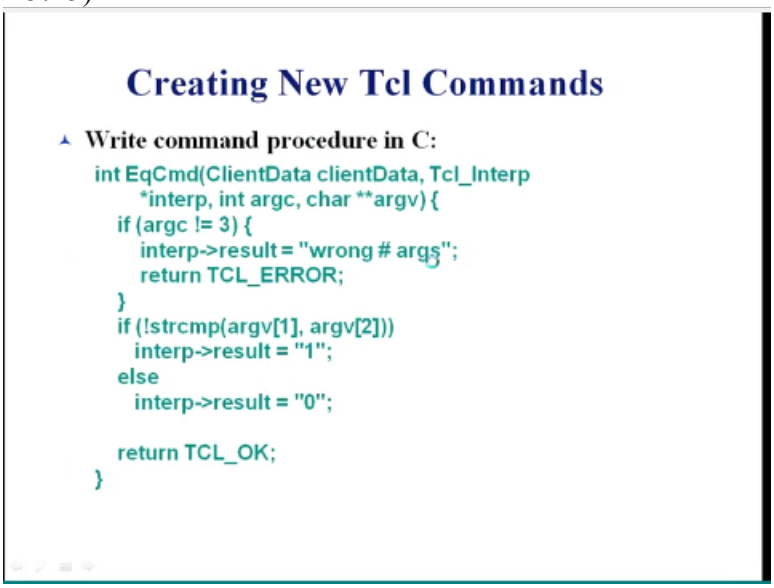
If it is a normal completion and then it is Tcl error,if any error occurred and the result during the execution, we can also point to the result to the inter arrow, to result and that points to the string ,that is whether it is Tcl okay or Tcl error and usually what we want is application should display the result or message for the user .

(Refer Slide Time: 24:50)



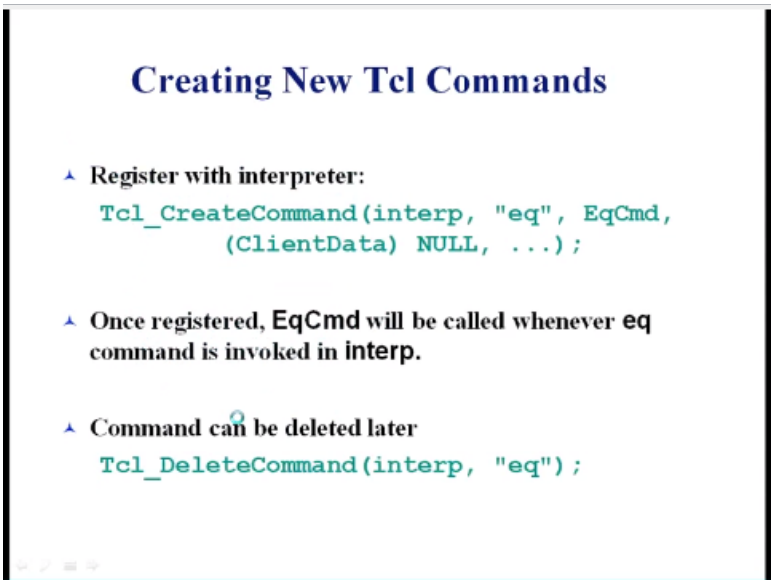
Now just to another thing is where do these script come from , read from the standard input I think I mean you can answer all , how do we read from the standard input ,read from the script file associated with x events, wait for events, invoke associated scripts and then embed it in c code.

(Refer Slide Time: 25:25)



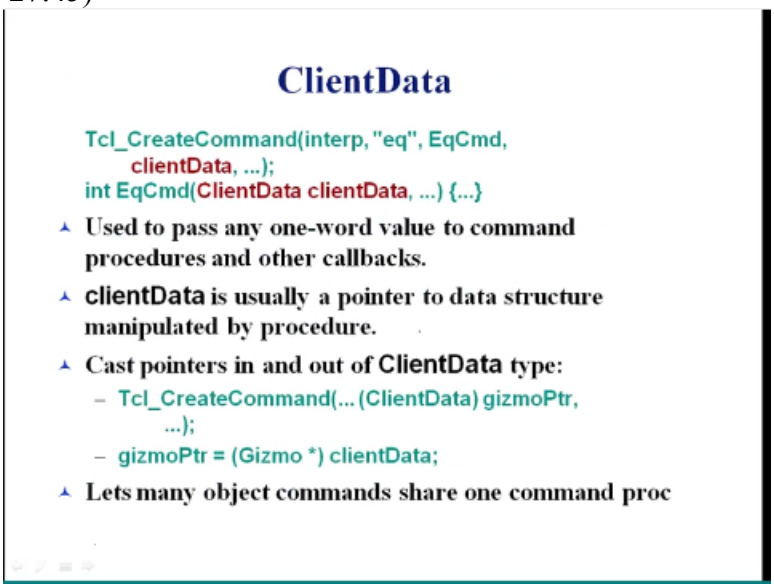

So here is a one way to create ,new commands ,so we call this Eq command as an integer and here ,we specify like client data, the Tcl interpreter and some argc and char argv, so if the arg c is not equal , basically we just throw the result as wrong number of arguments and then return error, now the next one, that we will see is, we do a string comparison, between the data 1 and data 2 and if they are equal then, we say basically that the commands are equivalent , otherwise we will say the command is not equal and then we just return the Tcl ok value.

(Refer Slide Time: 26:35)



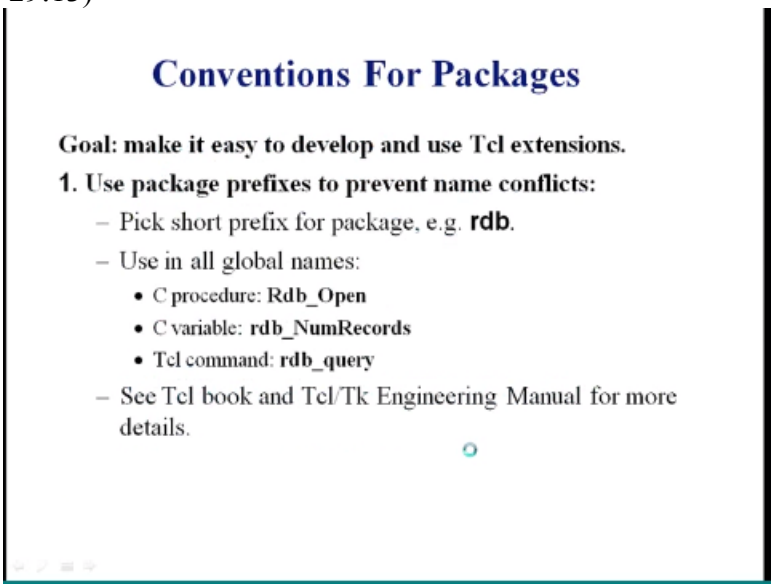

So now, to register with interpreter ,we need to do the Tcl create command interpreter eq ,Eq command, client data and null ,so once we register then the command will be called whenever ,any eq is invoked in the interpreter ,so we just kind of a binding eq to each command, so that if we just type eq the interpreter knows that it is command and we can also delete the command to this another function, which is Tcl delete command , so the Tcl create command ,creates the shortcut and then Tcl delete command will delete that .

(Refer Slide Time: 27:45)




So here is the client data essentially , so the create command ,we create that client data and then ,we say like Eq command client data, client data, so you usually the pass only one word value to the commands, procedures and their callbacks, but the client data here is usually a

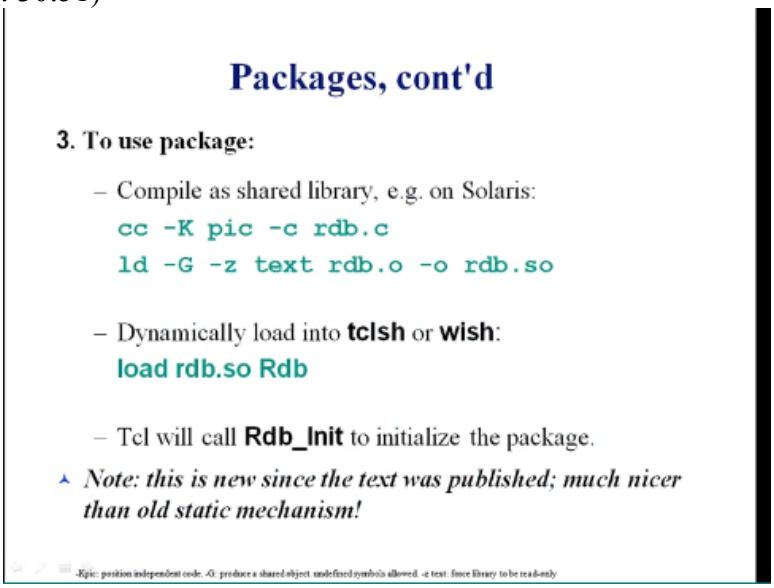
pointer to a data structure and manipulate by the procedure ,so we can cast the pointers in and out of the client data type, basically, for example like Tcl create demo, client data .
And then disappointed, at the give all pointer ,we can define this Gizmo* and then client data , so essentially it lets many object commands and share one command procedure.
(Refer Slide Time: 29:13)




So what are the conventions ,for defining packages in Tcl, so one is the goal is to make it easy ,to improve and use the Tcl extension ,so we can use the package prefixes to prevent the name conflicts ,so we basically like kind of pick a short prefix for or a package , for example the rdb is read and then we use, in all global names, so C procedure will have rdb open, C variable will have rdb num records and then Tcl command will have db query.
So this way like I mean, if we follow this ,definite of prefixes, then we can be successful in using Tcl.
(Refer Slide Time: 30:08)

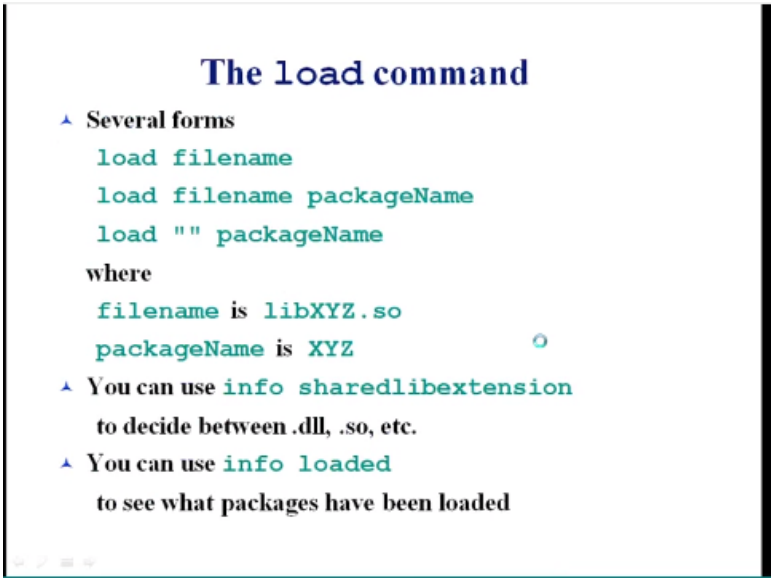

So here is another, example of creating the package initialization procedure ,name of the package is Rdb In it , the create packages commands and it creates the packages command and evaluates the startup script ,if there is any ,so how does it ,do it, so Rdb Init Tcl interpreter and then we will create Tcl commands one by one and then finally ,we will return the Tcl eval file .

(Refer Slide Time: 30:51)



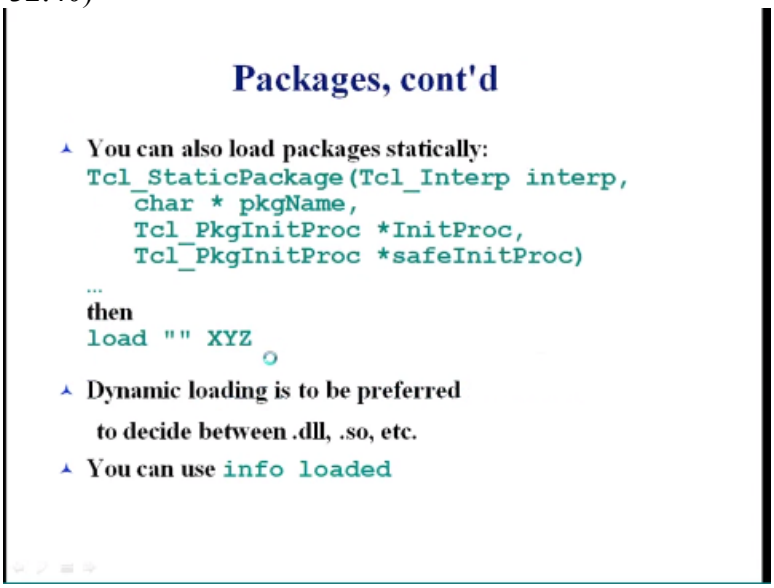

To use the package ,essentially like I will use this compile, as shared library for example Solaris, this is cc-K pic –c rdb.c and then we can also dynamically load Tcl programs, but mostly into tcl sh or wish, these are the two shells, that can load programs and then Tcl will call the Rdb Init to initialize package ,command for dynamically loading , thing is basically load rdb.so Rdb and then leaves it as rdb.

(Refer Slide Time: 31:44)



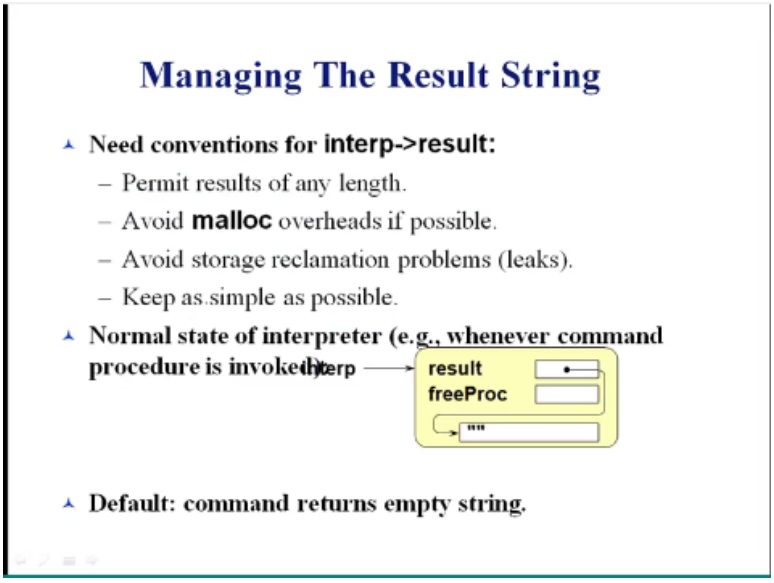

Now the load command, that can have several forms ,load file name ,load filename package name ,load may be file name and package name, that as the file name is , its tagged as lib XYZ. so, so it is the compiled form of the library and then the package name is XYZ ,which and we can use the info shared life extension to decide, between .dll or .so etc, we can also use info loaded command, to see what packages have been loaded, if they are active .

(Refer Slide Time: 32:40)




You can also load packages statically , that is Tcl static package, Tcl interpreter character ,what is the character and then we basically the package init , and then we just say load with an empty string XY .

(Refer Slide Time: 33:27)




So managing the results string, so we need conventions for interpreter result or basically how the result is , so one thing is it ,permits the result of any length, so and then we have to avoid malloc overheads, if possible and then avoid storage, reclamation problems and then keep it as simple as possible, so the last one is earlier said ,than done ,but that is what and then finally then and then the normal state of the interpreter is to invoke, the inter interpreter essentially.

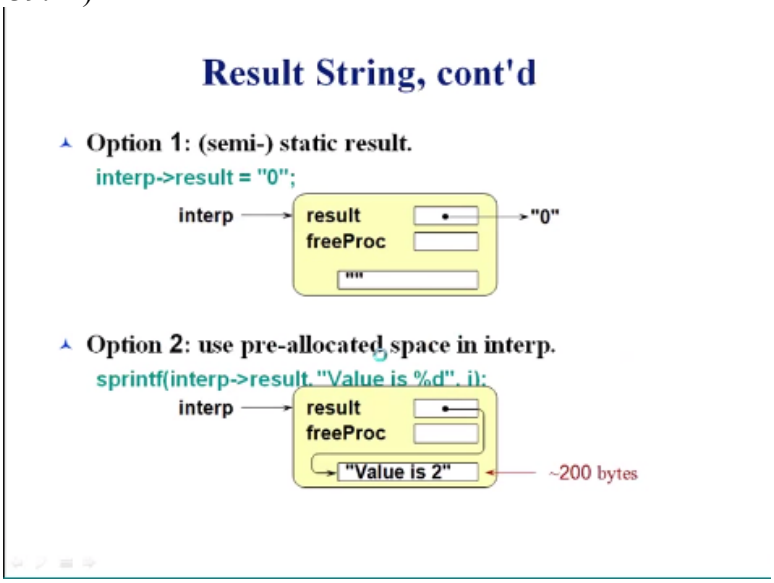
So here basically like the result have some number here and then it is called processor okay, so as I mentioned we can register the interpreter and then we can use the command ,as without pointing to any other place and then for the client data , essentially and then you can actually cast pointers in and out of client data type, by using these kind of commands ,actually like the create command, where you are inside that and because there is no pointer.

And then we define the gizmo pointers outside ,so it is class and then it also belongs to the client data, so the client data is usually a pointer ,to the data structure , so that is why and then the mini object-oriented of the object commands share one common problem, so we talked about this, we talked about these things and then here we talked about this dynamic loading, so the load command ,the load command is essentially like in many forms .

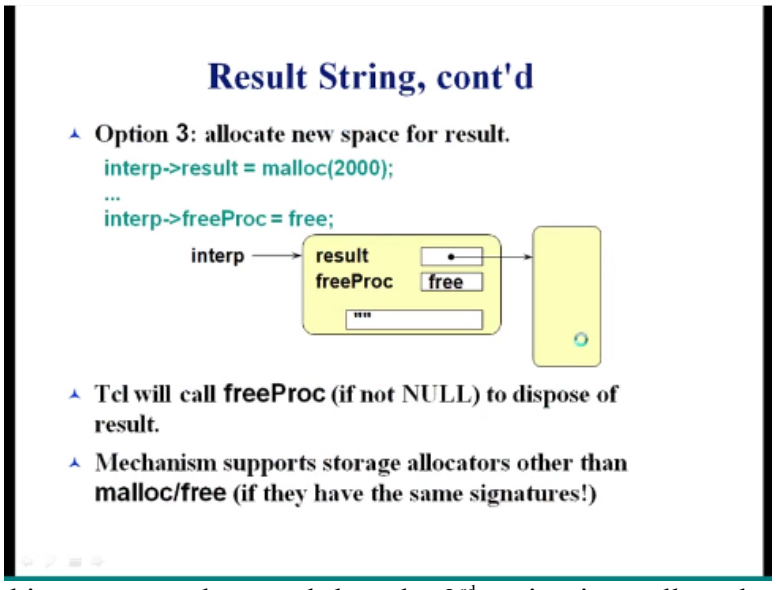
One is load test file name, we can also specify file name and then the package name or load everything in the package names, and the file name is lib XYZ. so, the package name is just XYZ , you can also in use the info shared library extension, for decide between dll ,so etc ,we can also use info loaded to see ,which package packages are loaded in the system , and also we can load the packages statically that is Tcl static package .

Tcl interpreter the characters in the package name ,then Tcl package in its proc basically , and it is s also using the safe init proc and if these conditions are met ,then something that takes YZ

which is one other thing and then the dynamic loading is always the poor, many tools, we saw this essentially like I means ,we need conventions for the interpreter followed by interpreter, result here we want to avoid malloc overhead.
So malloc is a sequence ,in memory and you do not want to, use that if possible and then we can avoid the storage within machine problems that is the leaks essentially, and then we want to keep the name as simple as possible ,so here one quick example this is the normal state of the interpreter ,that is the command procedure is not involved, what is the interpreter state ,so it writes out the results and then it is a free of the processes ,so by default the command returns empty string.
(Refer Slide Time: 39:14)

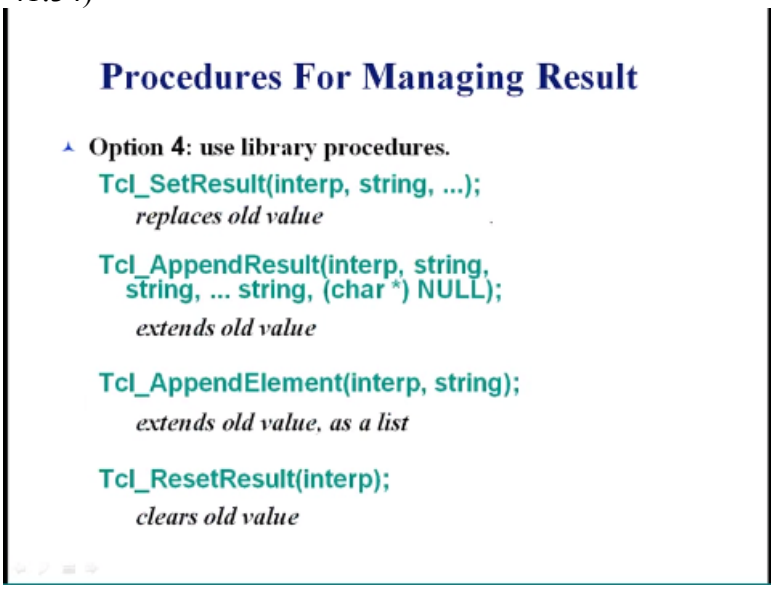



So a static ,semi static result is option 1,there we can write out ,saying that the result is 0 and that is about it , option 2 is the pre allocated space, in the interpreter, and then we can write it into that ,so sprint for a sprint of printers it is an interpreter result ,now is value is % d and then I, so that results in this kind of where, we have the result at the pointer here and then free proc value for that .
(Refer Slide Time : 40:52)



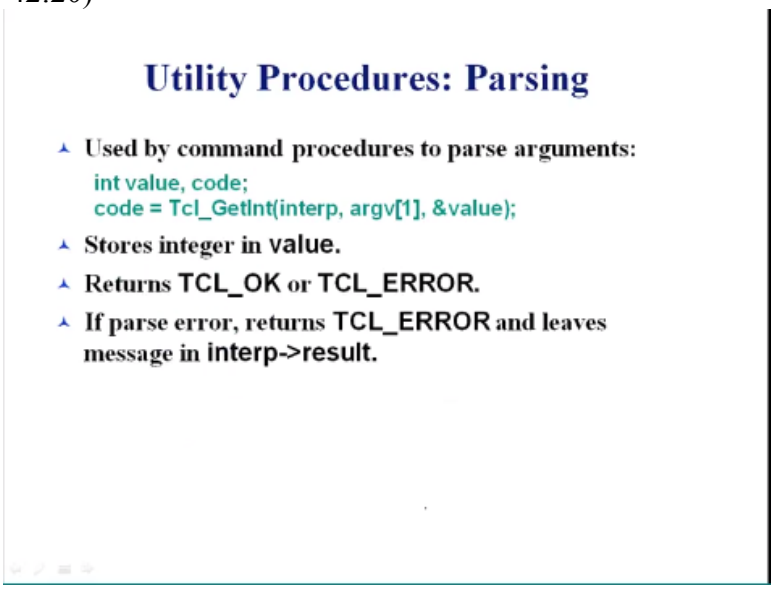

Even to all these things now, so here and then the 3rd option is to allow the new space for the result, which is the interpreters result =malloc (2000), then interrupt free proc is free and then the interpreter can put that as thing which is exhaust space, Tcl will call this free proc is not null , to dispose of the result ,so the result is created and then it is sent to the program for further processing ,while this program itself, so the mechanism ,supports storage allocators, other than malloc and free .

(Refer Slide Time: 41:34)

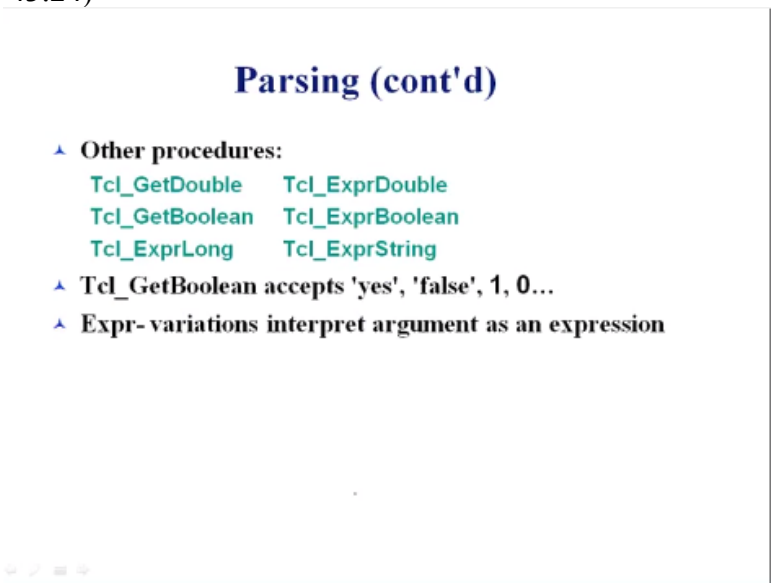



So how do we manage results ,essentially maybe we can use the library procedures , Tcl result set result is in the string and then we replace ,with the old value , Tcl append end result interpreter string, string, string and then char * null ,this extends the old value ,Tcl append

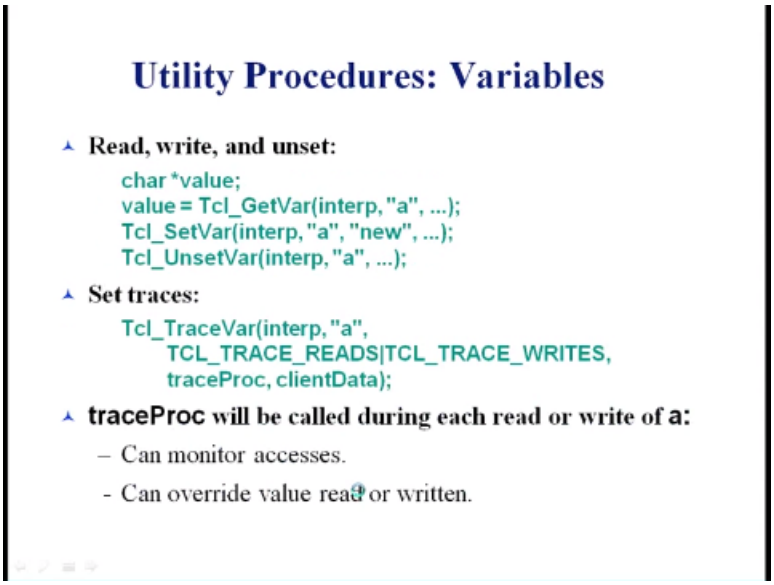
element and then interpreter ,when string extends the old value of a list and then finally there is a reset result ,which will just get a old value.
(Refer Slide Time: 42:20)




So the utility parsing is essentially like an command procedure, parse arguments, so here in integer, this is the one type of thing ,that code = Tcl get int interpreter argv 1 and value and it stores integer values ,return Tcl ok or Tcl error and then if there is a parse error.
(Refer Slide Time: 43:24)



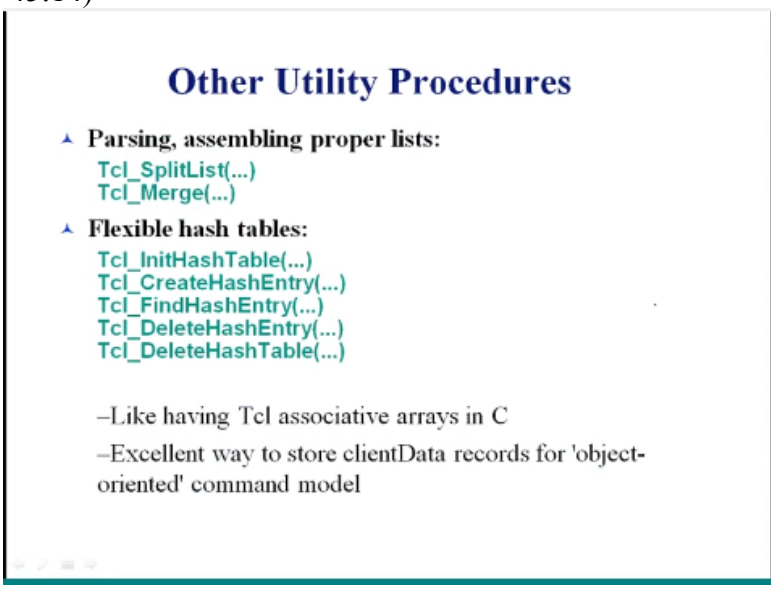

And then for parsing there are other procedures, Tcl get double, these are like double in the context of floating point numbers, Tcl expression double , get Boolean double ,expr Boolean and then expr long and expr string, Tcl get Boolean accepts yes , false ,1 or a zero , expr is basically like the variations of interpreter argument as an action .
(Refer Slide Time: 43:59)




So some more thing on the variables, read, write and unset, so how do we, do it is basically you insert the value and then value = Tcl get var interpreter a, whatever the value from, that get it and then we just set the var to a, a to the interpreter and then we unset for any of the other variables, formidable and then we can set traces basically , like I mean that is going from maybe various things essentially.

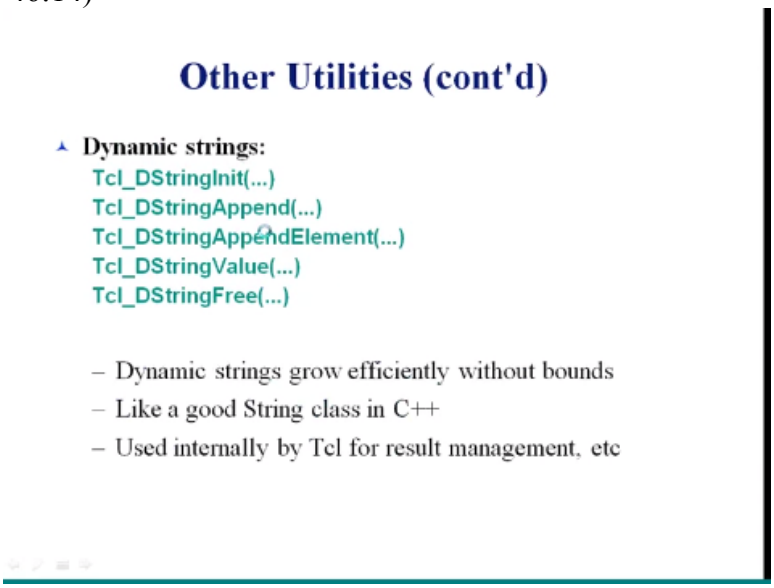
So interpreter quotes a Tcl trace reads that enables the Tcl trace reads and then Tcl trace write, and then trace proc and the command and the client data, so the trace proc will be called during each read or write of a monitor and then it can write any overriding parse at read or written .

(Refer Slide Time: 45:14)

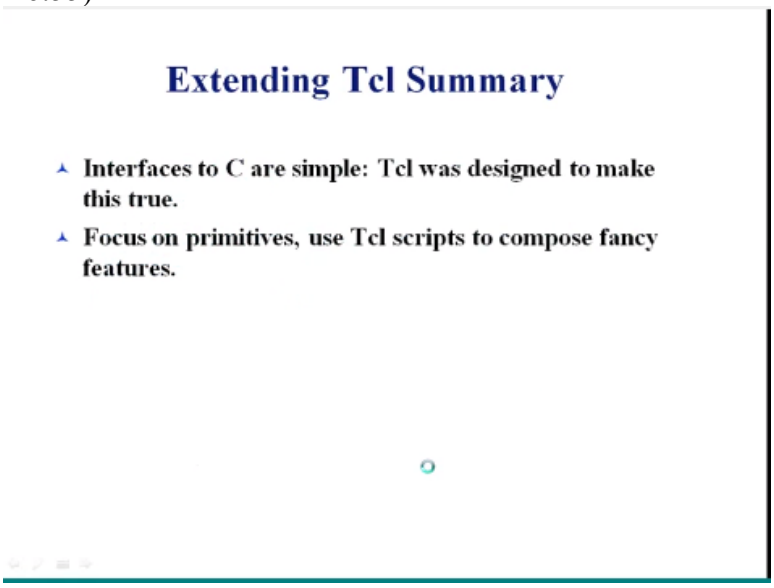



In the parsing assembling proper lists , like this basically is one split list and then merge ,but the more importantly like the dynamic another one like , split list essentially like list, splits all these

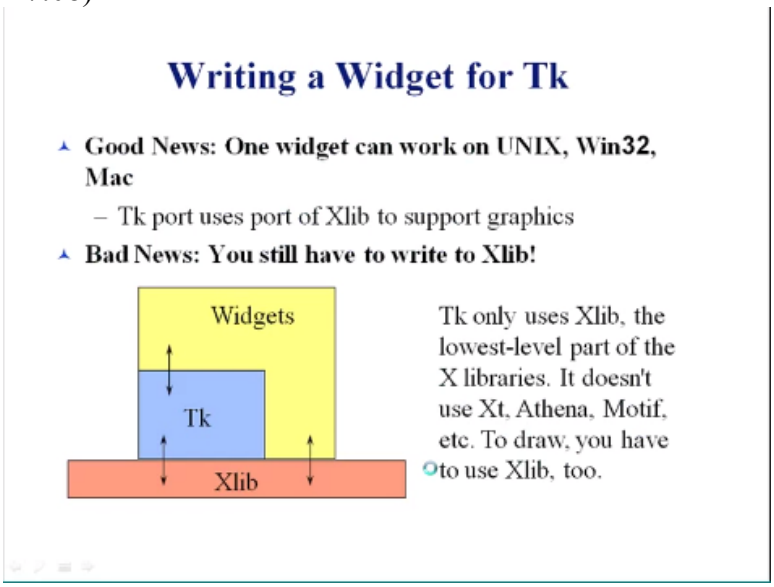
lists ,single multiple objects , Tcl merge command actually merges a list files, into a single data structure and then some flexible hash tables are initialized, hash table within it create hash entry and delete hash entry and delete hash table.
So these kinds of procedures are like having associative arrays in c, it will store the client data, records or object-oriented command.
(Refer Slide Time: 46:14)




Now comes through dynamic strings essentially, these are strings that can be handled dynamically, so D string in it , they also have a D string append ,this string append element this in value and then d string free, so dynamic string, a string can actually grow efficiently without bonds, between like a good string class in C++ and used internally by Tcl for result management etc.
(Refer Slide Time: 46:55)



I think I am going to stop, at this point ,we will continue extending the Tcl ,actually like I will just summarize, this one section now.
(Refer Slide Time: 47:08)




We will see the next section, in the next class, so finally on the summary essentially like interfaces to C are fairly simple, Tcl was designed to make this true, so has to be simple and then we want to focus on primitives ,use Tcl scripts to compose fancy features ,so today we saw like the couple of big topics ,one is we finished ,the canvas widget using and when demonstrated using, a furniture mover program and then on the second part,.
We actually went through how to interface ,with C language, so how to extend Tcl in to C, so I think that is pretty much , it we will cover some more items in the next one and hopefully I will wrap up next session, we need one more session to wrap up okay thank you very much .