

TASH
(Tcl Ada SHell)
An Ada binding to Tcl/Tk

SIGAda 2000
November, 2000

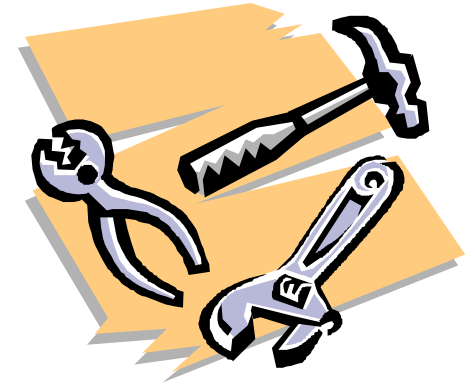
Terry Westley
<http://www.adatcl.com>

Tutorial Outline

4 Introduction to Tcl/Tk and TASH

- Scripting in Ada with TASH
- GUI programming in Ada with TASH

What is Tcl/Tk?



- **Tool Command Language**
 - Simple, powerful, and extensible scripting language
 - Compare to Bourne shell, C shell, Perl, awk
- **ToolKit**
 - Portable GUI toolkit extension of Tcl

Why I Like Scripting Languages

- Rapid development of small applications
- Integration of several small applications into one larger coherent application
- Operating system interfaces
- Text processing facilities

Scripting Languages

Capabilities

- Regular expression pattern matching and string substitution
- Associative arrays
- File and directory handling
- Execute another program
 - Send input to program
 - Read its output

Why Tcl/Tk in Particular?

- Platform independent
 - Unix (X Windows), Windows, and Macintosh
- Embeddable
- Extensible
 - Object oriented
 - Widget libraries
 - Oracle interface
- Open Source

Sample Tcl Script

Identify Programming Language

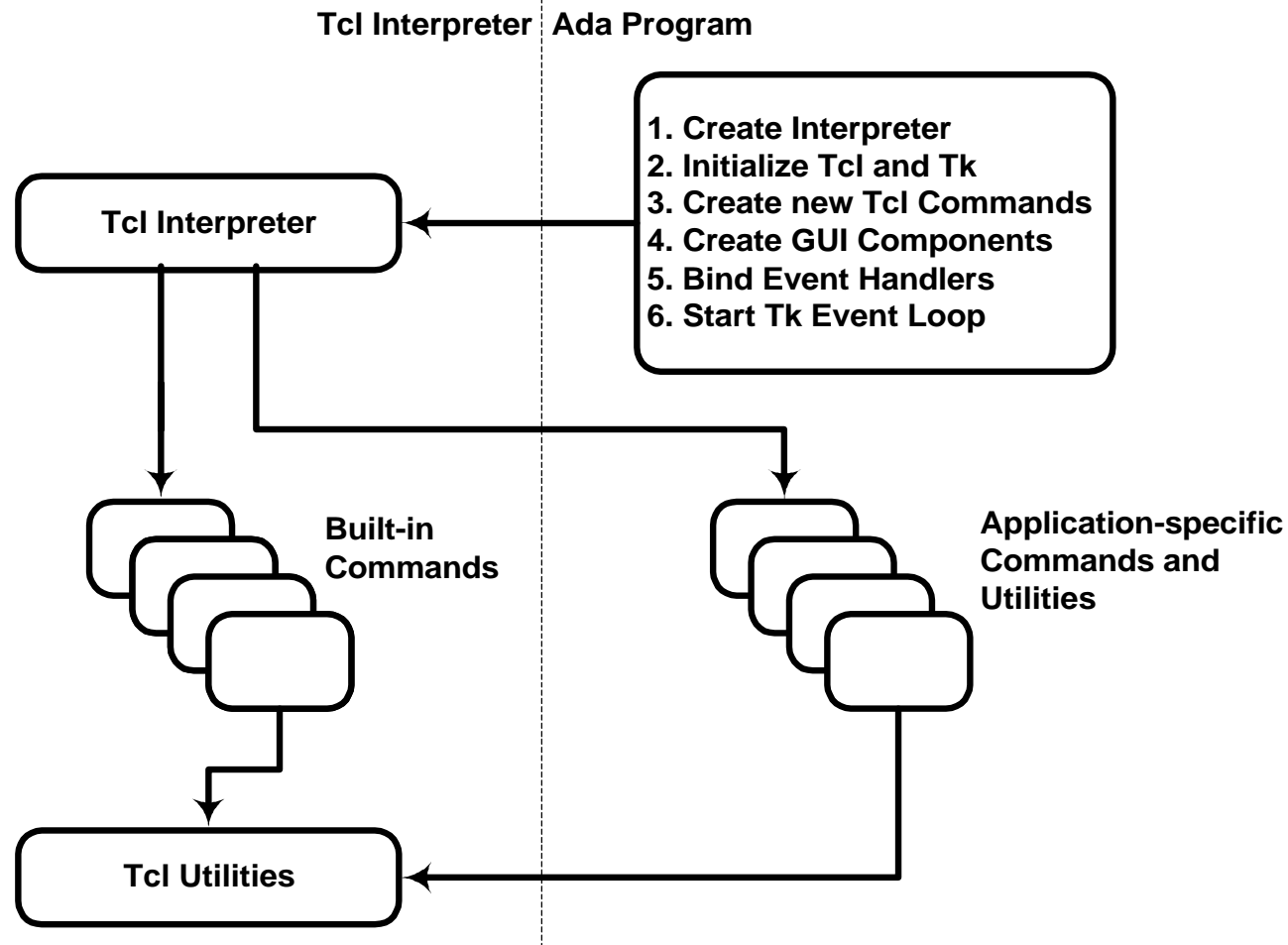
Command line arguments

[] -- execute a command

```
foreach file $argv {  
    set extension [file extension $file]  
    switch -glob $extension {  
        ".tcl"           {puts "$file tcl"}  
        ".ad[abs]"      {puts "$file ada"}  
        ".[ch]"          {puts "$file c"}  
        default          {puts "$file unknown"}  
    }  
}
```

"glob" style pattern matching

Tcl Interpreter Interaction (Embedded Tcl)



Why I Don't Like Scripting Languages

- Lack of strong typing
- Some scripting languages have very limited modularity facilities
- Lack of module interface checks
- Poor performance

What is TASH?

- Free Ada binding to Tcl/Tk
- TASH is **not** a scripting language with an Ada-like syntax

Why Use TASH?

- To extend Tcl by adding new commands coded in Ada instead of C
- To use Tcl capabilities in an Ada program
 - Regular expression pattern matching
 - Associative arrays
- Platform independence
 - Operating system interface
 - GUI toolkit

Sample TASH Program

```
for I in 1..Ada.Command_Line.Argument_Count loop
  declare
    File_Name : constant String := Ada.Command_Line.Argument (I);
    Extension : constant String := Tash.File.Extension (File_Name);
  begin
    if Extension = ".tcl" then
      Ada.Text_IO.Put_Line (File_Name & " tcl");
    elsif Tash.Regexp.Match (Extension, ".ad[sb]") then
      Ada.Text_IO.Put_Line (File_Name & " ada");
    elsif Tash.Regexp.Match (Extension, ".[ch]") then
      Ada.Text_IO.Put_Line (File_Name & " c");
    else
      Ada.Text_IO.Put_Line (File_Name & " unknown");
    end if;
  end;
end loop;
```

Ada Binding

- Thin binding
- Thinner binding
- Thick binding

Thinner Binding

- Implements Tcl APIs (tcl.h)
 - Subprograms and data types correspond one-to-one to C functions and Tcl data types
- Uses Tcl return codes
- Uses C data types (Interfaces.C)
- Tcl and Tcl.Tk packages

Thin Binding

- Implements Tcl APIs (tcl.h)
 - Again: Subprograms and data types correspond one-to-one to C functions and Tcl data types
- Uses exceptions in place of return codes
- Uses Ada data types
- Tcl.Ada and Tcl.Tk.Ada packages

Thick Binding

- Interface to Tcl capabilities
 - Not a one-to-one correspondance with C functions or Tcl data types
- Uses tagged controlled types
- Hides and protects access to Tcl interpreter
- Automatically initializes Tcl
- Tash package and its children



Just say NO

Tutorial Outline

- Introduction to Tcl/Tk and TASH
- 4** Scripting in Ada with TASH
- GUI programming in Ada with TASH

Scripting vs Ada

- Scripting good for
 - small applications (< 1 KSLOC)
 - where strings are chief data type
 - integration of several programs into one application
 - GUI front-end to command line program

Scripting vs Ada

- Ada much better for
 - larger applications
 - non-string and complex data types
 - higher performance

TASH Scripting Capabilities

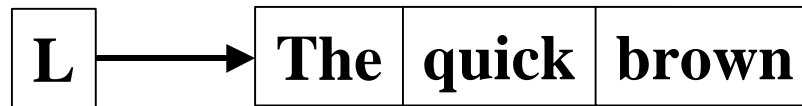
4List object

- Regular expression pattern matching
- Associative arrays
- File and directory handling
- File I/O
- Executing other programs
- C-style printf formatting
- Platform information

TASH List Object

- Based on Tcl lists
- Zero or more elements in an ordered list
- Each element may be a scalar type, a string or a list
- Generic list package for handling
 - integer types
 - floating point types
- Elements of different types may be stored in one list

Common List Functions



declare

L : Tash.Lists.Tash_List;

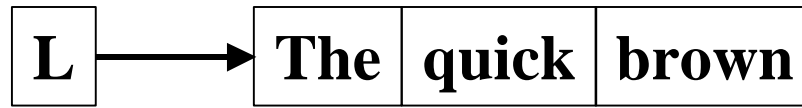
begin

L := Tash.Lists.To_Tash_List ("quick");

Tash.Lists.Append (L, "brown");

Tash.Lists.Insert (L, 1, "The");

Common List Functions



`Tash.Lists.Head (L) = "The"`

`Tash.Lists.Tail (L) = "brown"`

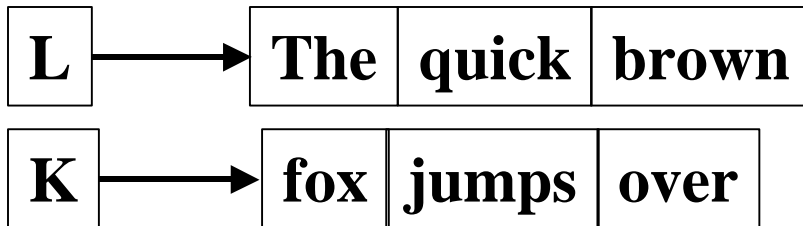
`Tash.Lists.Get_Element (L, 2) = "quick"`

`Tash.Lists.Length (L) = 3`

`Tash.Lists.Is_Empty (L) = False`

`Tash.Lists.To_String (L) = "The quick brown"`

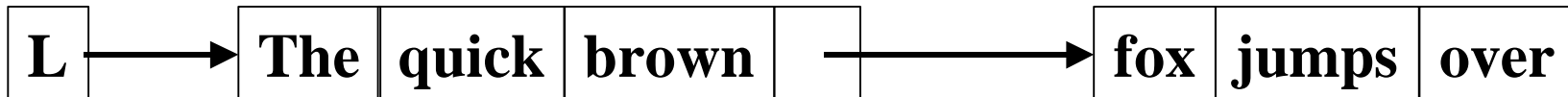
Append Elements vs List



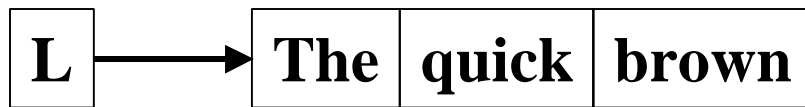
```
Tash.Lists.Append_Elements (L, K);
```



```
Tash.Lists.Append_List (L, K)
```



Concatenate List & String



J := L & K;



J := J & "the" & "lazy" & "dog";



Delete List Elements

```
Tash.Lists.Delete_Element (L, 3);
```

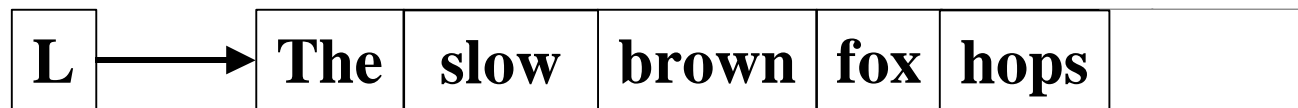
```
Tash.Lists.Delete_Slice (L, 2, 4);
```



Replace List Elements

```
Tash.Lists.Replace_Element (L, 2, "slow");
```

```
Tash.Lists.Replace_Slice (L, 5, 6, "hops");
```



Push and Pop

```
Tash.Lists.Pop (L);
```

```
Tash.List.Push (L, "A");
```



Sort A List

```
L := Sort (L);
```

```
L := Sort (L, Order => Decreasing);
```



Split a String to Create a List

```
L := Tash.Lists.Split (  
  Str      => "http://www.adatcl.com/docs/intro.htm",  
  Split_At => "/" );
```



Create a List from a String

```
L := Tash.Lists.To_Tash_List (  
    "{Terry Westley} twestley@acm.org");
```



Join the Elements of a List



```
Tash.Lists.Join (L, ",") =
```

```
"Terry Westley,twestley@acm.org"
```


Integer and Float List Elements



```
Tash.Lists.Replace_Element (L, 7, "one");
```

```
Tash.Integer_Lists.Replace_Element (L, 7, 1);
```

```
Tash.Float_Lists.Replace_Element (L, 7, 1.0);
```

TASH Scripting Capabilities

- List object
- Regular expression pattern matching
- Associative arrays
- File and directory handling
- File I/O
- Executing other programs
- C-style printf formatting
- Platform information

Regular Expression Query

```
Tash.Regexp.Match (  
    Source  => "Tcl/Tk",  
    Pattern => "T[cC]") = True  
Tash.Regexp.Match (  
    Source  => "Tcl/Tk",  
    Pattern => "(Ada|Tk)") = True
```

- For complete details on forming regular expressions, see Tcl documentation page *re_syntax* or *IEEE 1003.2*

Regular Expressions

- Atom
 - A character
 - A character group, e.g. [a-zA-Z], [cC], [^0-9]
 - Any single character: .
 - A branch
- Quantifier
 - * = sequence of 0 or more atoms
 - + = sequence of 1 or more atoms
 - ? = 0 or 1 atom
- Branch
 - Match one of several patterns: (Tcl|Tk)

Regular Expression Examples

- An Ada comment line:

$\wedge *_{--}+$

- An Ada identifier:

$[A-Z][A-Za-z0-9_]*$

- A floating point number:

$(+|-)?[0-9]+\.[0-9]+([eE](+|-)?[0-9]+)?$

Regular Expression Example

$(+|-)? [0-9]+ \backslash. [0-9]+ ([eE](+|-)?[0-9]+)?$

Sign: 0 or 1 of either + or -

Fore: 1 or more digits

Decimal point

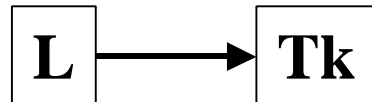
Aft: 1 or more digits

Exponent: 0 or 1 of:

e or E followed by an optional sign followed by 1 or more digits

Extract a Substring

```
L := Tash.Regexp.Match (  
  Source  => "Tcl/Tk",  
  Pattern => "T[kK]$" );
```



Match and Extract Multiple Substrings

```
L := Tash.Regexp.Match (  
  Source  => "Tcl/Tk",  
  Pattern => "(Tcl)/(Tk)");
```



String Substitution

```
Tash.Regexp.Substitute (  
    Source      => "Tcl/Tk",  
    Pattern     => "(Tcl/|Tk)*",  
    Sub_Spec    => "Use TASH" ) =  
  
    "Use TASH"
```

Other Regular Expression Features

- Ignore case in source string
- Search for strings in the elements of a list
 - Going forward
 - Going backward

TASH Scripting Capabilities

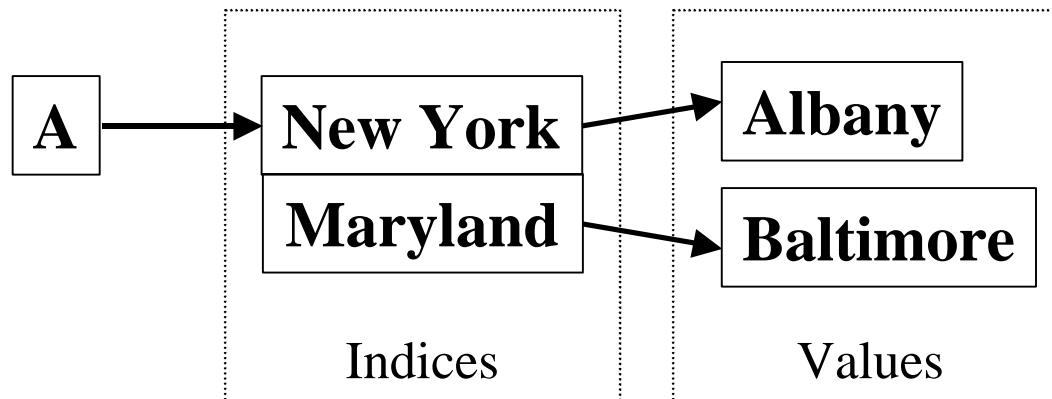
- List object
- Regular expression pattern matching
- Associative arrays
- File and directory handling
- File I/O
- Executing other programs
- C-style printf formatting
- Platform information

Associative Arrays

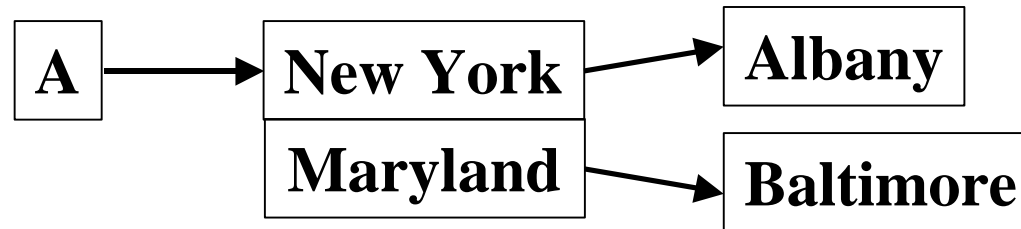
- Based on Tcl arrays
- Zero or more elements each indexed by a string
- Each element value may be a scalar type, a string or a list
- Generic array packages for handling
 - integer types
 - floating point types
- Element values of different types may be stored in one array

Create an Array

```
declare
  A : Tash.Arrays.Tash_Array;
begin
  A := Tash.Arrays.To_Tash_Array (
    "{New York} Albany Maryland Baltimore");
```



Get Array Indices



```
L := Tash.Arrays.Get_Indices (A);
```



```
L := Tash.Arrays.Get_Indices (A, "Mary*");
```



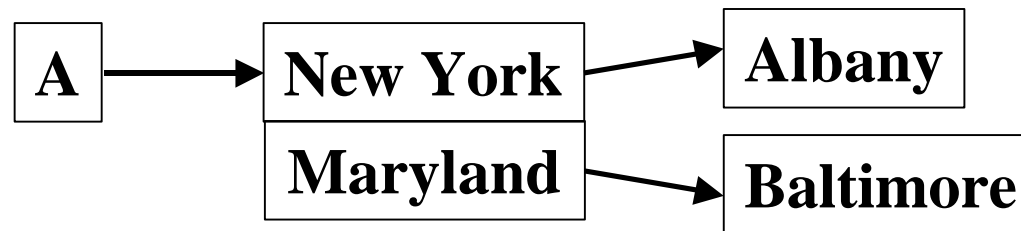
Glob-Style Pattern Matching

- Not full regular expressions
- Match any single character: `?`
- Match any sequence of zero or more characters: `*`
- Character group: `[A-Z]`, `[0-9]`
- Alternation: `{a,b,...}`
- For complete details on forming glob-style patterns, see Tcl documentation page *glob*

RE vs Glob Patterns

- Regular Expressions are more powerful, resulting, often, in more complicated patterns
- Regular Expressions can match a portion of a string
- Glob-style patterns usually must match the whole string

Get Array Indices



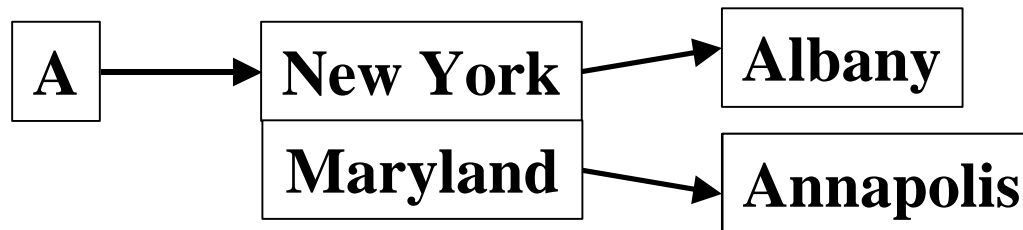
```
Tash.Arrays.Get_Indices (A) =  
    "{New York} Maryland"
```

```
Tash.Arrays.Get_Indices (A, "Mary*") =  
    "Maryland"
```

```
Tash.Arrays.Get_Indices (A, "*[Yy]*") =  
    "{New York} Maryland"
```

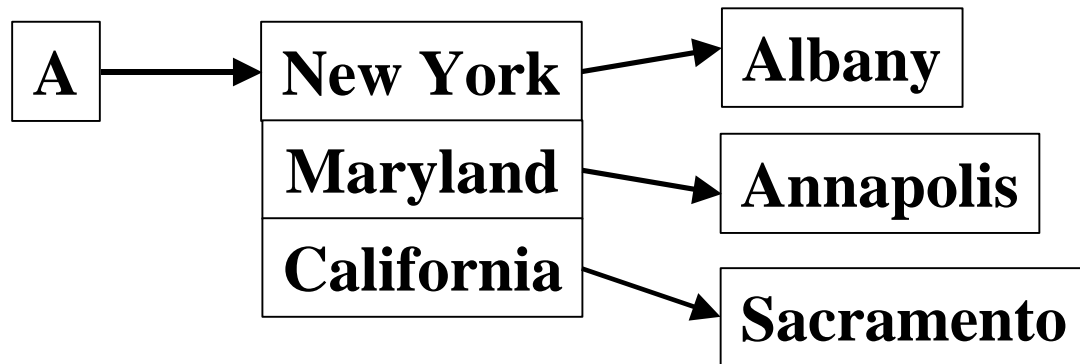
Set Array Element

```
Tash.Arrays.Set_Element (  
    TArray => A,  
    Index  => "Maryland",  
    Value  => "Annapolis");
```



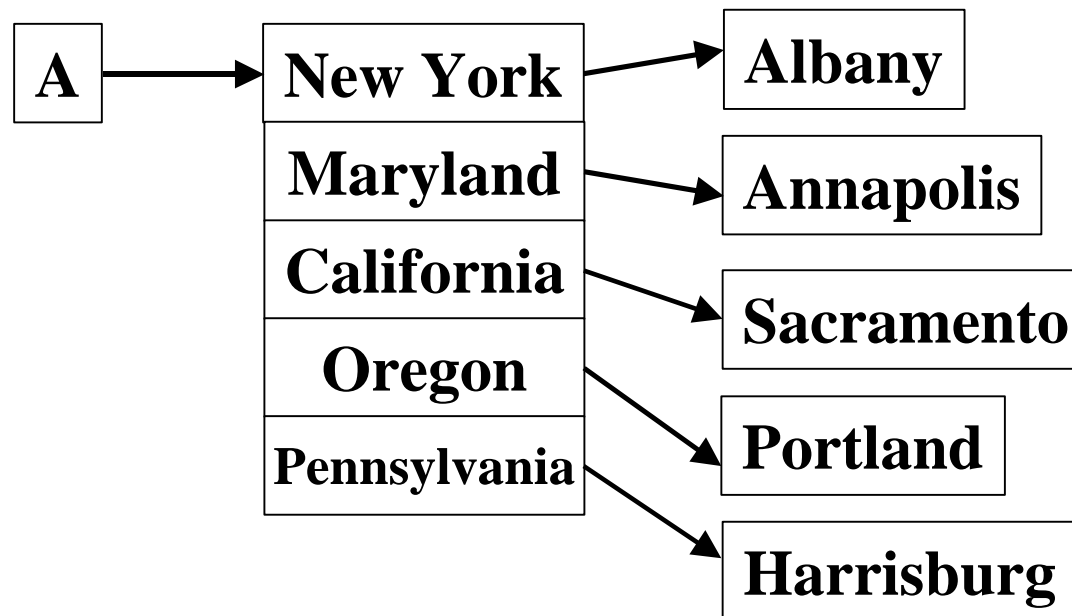
Set Array Element

```
Tash.Arrays.Set_Element (  
    TArray => A,  
    Index  => "California",  
    Value  => "Sacramento");
```



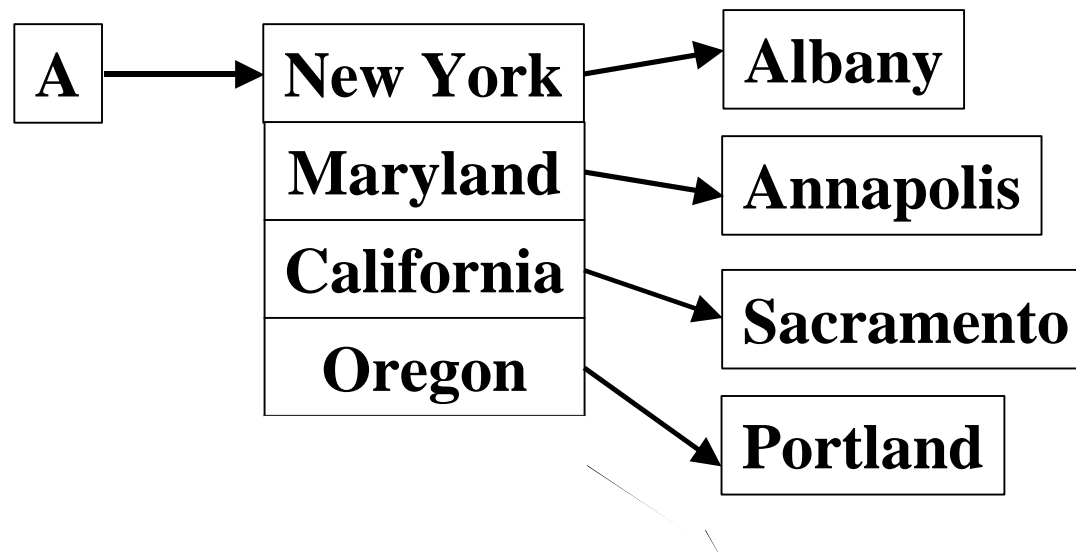
Set Array Elements

```
Tash.Arrays.Set_Elements (A,  
    "Oregon Portland Pennsylvania Harrisburg");
```

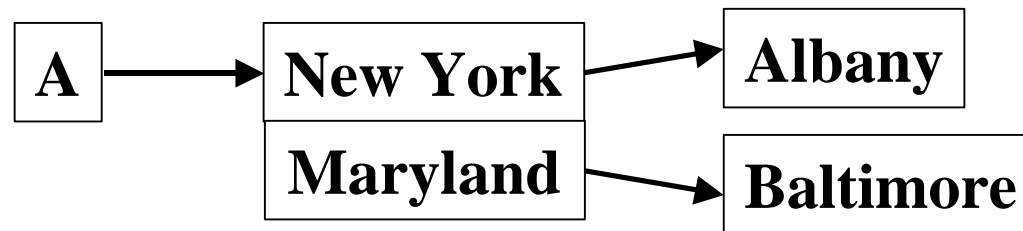


Delete Element

```
Tash.Arrays.Delete_Element (A, "Pennsylvania");
```



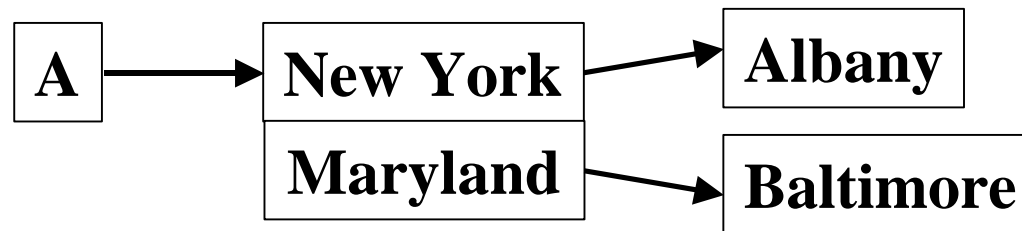
Get Elements Sorted by Indices



```
L := Tash.Arrays.Get_Sorted_Elements (A);
```



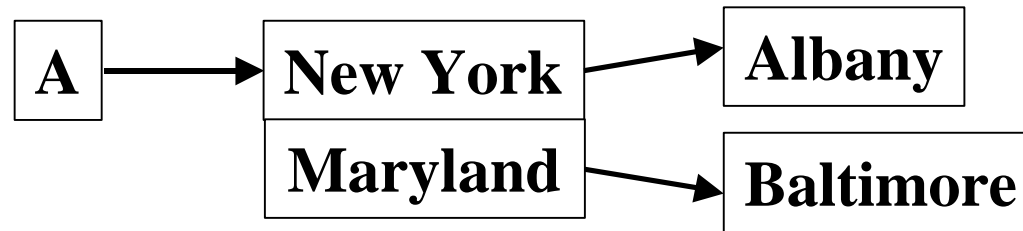
Get Sorted Indices



```
L := Tash.Lists.Sort (  
    Tash.Arrays.Get_Indices (A));
```



Array Query Functions



```
Tash.Arrays.Is_Empty (A) = False
```

```
Tash.Arrays.Length (A) = 2
```

```
Tash.Arrays.Get_Element (A, "Maryland") = "Baltimore"
```

```
Tash.Arrays.Get_Elements (A, "Mary*") = "Baltimore"
```

```
Tash.Arrays.Get_Elements (A, "*[Yy]*") =  
    "Albany Baltimore"
```

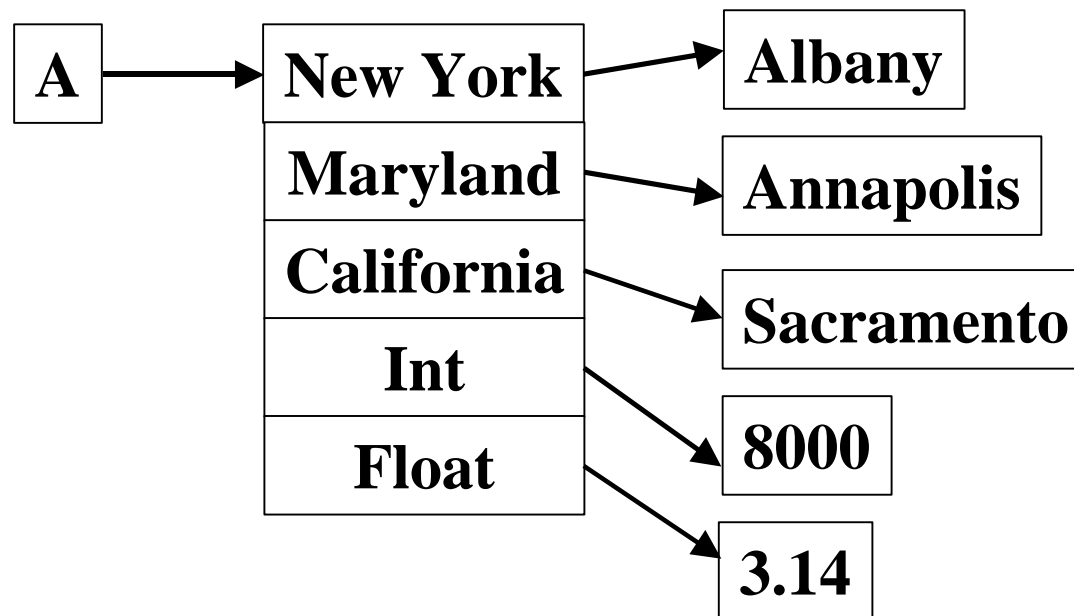
```
Tash.Arrays.Exists (A, "Colorado") = False
```

```
Tash.Arrays.To_String (A) =  
    "{New York} Albany Maryland Baltimore"
```


Integer and Float Array Values

```
Tash.Integer_Arrays.Set_Element (A, "Int", 8000);
```

```
Tash.Float_Arrays.Set_Element (A, "Float", 3.14);
```



TASH Scripting Capabilities

- List object
- Regular expression pattern matching
- Associative arrays
- File and directory handling
- File I/O
- Executing other programs
- C-style printf formatting
- Platform information

Get and Set

File Access and Modified Times

```
Tash.File.Get_Access_Time ("tash-file.ads")  
Tash.File.Set_Access_Time ("tash-file.ads",  
    Calendar.Clock);  
Tash.File.Get_Modified_Time ("tash-file.ads")  
Tash.File.Set_Modified_Time ("tash-file.ads",  
    Calendar.Clock);
```

Get and Set File Attributes

```
Tash.File.Get_Attribute (  
    "tash-file.ads", ShortName)
```

```
Tash.File.Set_Attribute (  
    "tash-file.ads", Hidden, "0");
```

File Attributes

Group	Unix	group name of a file
Owner	Unix	user name of the owner of a file
Permissions	Unix	permissions as octal code used by chmod(1)
Archive	Win	archive attribute of a file ("0" or "1")
Hidden	Win/Mac	hidden attribute of a file ("0" or "1")
LongName	Win	expands each path element to its long version (this attribute cannot be set)
ReadOnly	Win/Mac	readonly attribute of a file ("0" or "1")
ShortName	Win	returns string where each path element is replaced with its short (8.3) version of the name (this attribute cannot be set)
SystemAttr	Win	system attribute of a file ("0" or "1")
Creator	Mac	Finder creator type of the file
Ftype	Mac	Finder file type of file

Extract File Name Portions

```
Tash.File.Directory_Name ("/usr/lib/libm.a") =  
    "/usr/lib"
```

```
Tash.File.Extension ("/usr/lib/libm.a") =  
    ".a"
```

```
Tash.File.Root_Name ("/usr/lib/libm.a") =  
    "/usr/lib/libm"
```

```
Tash.File.Tail name ("/usr/lib/libm.a") =  
    "libm.a"
```

File and Directory Query Functions

- Determine if a file or directory
 - Is a regular file
 - Is executable
 - Exists
 - Is a directory
 - Is readable by user
 - Is writable by user
 - Is owned by user
 - Is a link

File and Directory Query Functions

- Get native name
- Get path type (absolute, relative, or volume relative)
- Get file name a link references
- Get size of a file
- Split a file name at its path separators

Get File Type

- File
- Directory
- CharacterSpecial
- BlockSpecial
- Fifo
- Link
- Socket

File and Directory Query Functions

- Get volume names
 - Unix: always ("/")
 - Windows: ("a:/", "c:/", "d:/")
- Get current working directory

File Name Pattern Matching

```
function Match (  
    Pattern          : in String;  
    Directory        : in String := "";  
    Path_Prefix      : in String := "";  
    Type_List        : in String := "")  
return Tash.Lists.Tash_List;
```

```
Tash.File.Match ("*", Directory => "../bin")
```



File and Directory Modification Functions

- Copy files and directories
- Delete files and directories
- Create a directory
- Rename a file or directory

Other File and Directory Functions

- Join strings to form a file name with proper platform-dependent path separator
- Change current working directory

TASH Scripting Capabilities

- List object
- Regular expression pattern matching
- Associative arrays
- File and directory handling

4File I/O

- Executing other programs
- C-style printf formatting
- Platform information

TASH File I/O

- Open
 - File
 - Serial port
 - Command pipeline
- Close
- Flush
- Get list of process IDs of a command pipeline

TASH File I/O

- New Line
- Get, Get_Line
- Put, Put_Line
- Get blocking mode
- Get and set buffering mode
- Get and set buffer size
- Get and set translation mode

TASH Scripting Capabilities

- List object
- Regular expression pattern matching
- Associative arrays
- File and directory handling
- File I/O
- Executing other programs
- C-style printf formatting
- Platform information

Command Pipeline

```
declare
  Pipe : Tash.File_IO.File_Type;
  Line : String (1..1000);
  Last : Natural;
begin
  Tash.File_IO.Open (
    File => Pipe,
    Name => "| ps -ef | grep -i oracle",
    Mode => Tash.File_IO.Read);
  while not Tash.File_IO.End_Of_File (Pipe) loop
    Tash.File_IO.Get_Line (Test_File, Line, Last);
    Ada.Text_IO.Put_Line (Line (1..Last));
  end loop;
  Tash.File_IO.Close (Pipe);
end;
```

TASH Scripting Capabilities

- List object
- Regular expression pattern matching
- Associative arrays
- File and directory handling
- File I/O
- Executing other programs
- C-style printf formatting
- Platform information

C-Style Printf Formatting

```
Tash.Lists.Format (  
    "The result for %-15s is %5.2f (%04x)",  
    To_Tash_List("{a piece of pi}") & 3.14159 & 89) =  
  
    "The result for a piece of pi    is  3.14 (0059)"
```

TASH Scripting Capabilities

- List object
- Regular expression pattern matching
- Associative arrays
- File and directory handling
- File I/O
- Executing other programs
- C-style printf formatting

4 Platform information

Platform Information

```
Tash.Platform.Byte_Order =  
    "littleEndian" or "bigEndian"  
Tash.Platform.Machine =  
    "intel", "PPC", "68k", or "sun4m"  
Tash.Platform.OS =  
    "Windows 95", "Windows NT", "MacOS", or "SunOS"  
Tash.Platform.OS_Version      =  
    version number of the operating system  
Tash.Platform.Platform =  
    "windows", "macintosh", or "unix"  
Tash.Platform.User_Name =  
    identifies current user
```

Tcl Capabilities Missing from TASH Thick Binding

- Windows registry editing
- Windows Dynamic Data Exchange (DDE)
- Communication protocols
 - TCP, UDP, HTTP
- Manipulate binary data
- Time functions
- File event handlers
- Load binary library or Tcl package

Tutorial Outline

- Introduction to Tcl/Tk and TASH
- Scripting in Ada with TASH
- 4** GUI programming in Ada with TASH

Ada GUIs with TASH

- TASH provides an alternative to using platform-dependent GUI APIs
 - Windows API for GUI development on Windows 95/98/Me/NT/2000
 - X Window system on Unix

Sample GUI Application

A simple GUI application for computing future value of a series of fixed monthly investments will be demonstrated

$$\text{Future Value} = M * \frac{(1+i)^n - 1}{i}$$

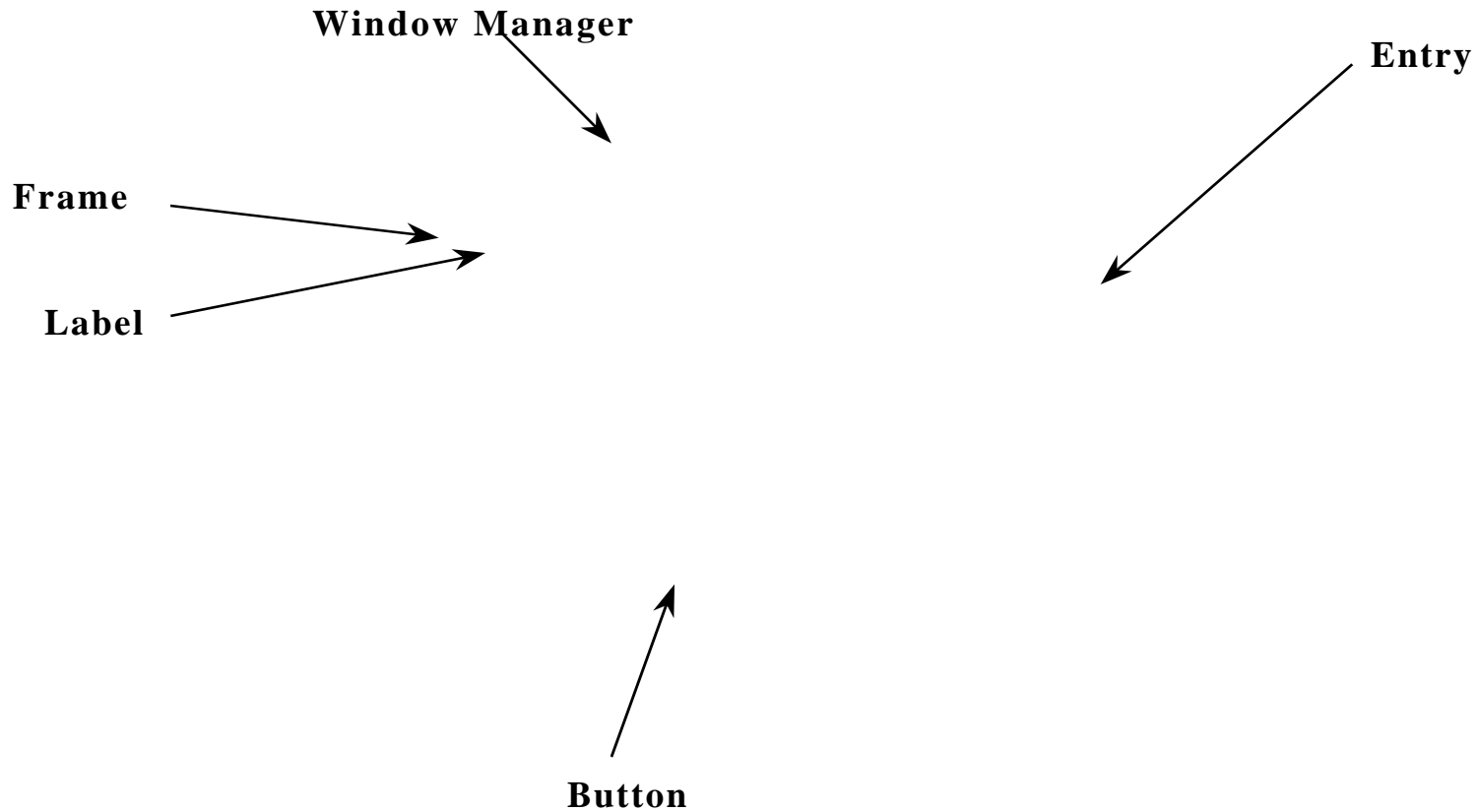
where M = monthly investment
 i = interest rate per month
 n = number of months

Sample Screenshot

Pattern of a TASH Program

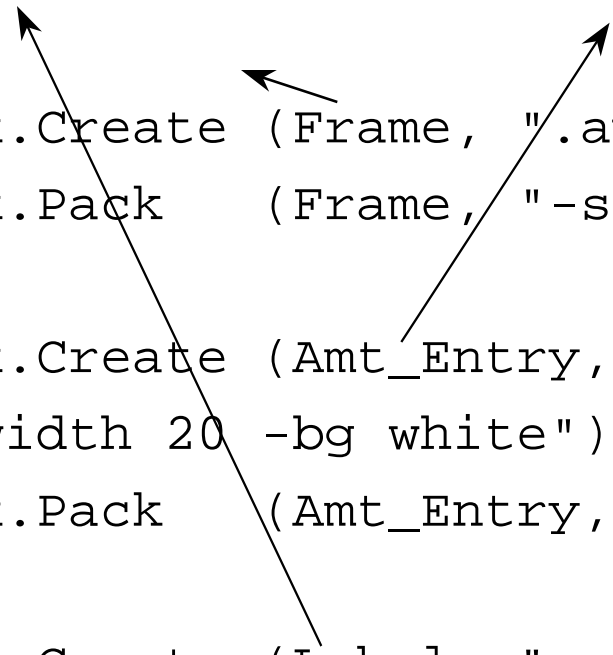
- Start Tcl Interpreter
- Initialize Tcl and Tk
- Create new Tcl commands
- Create GUI components
- Bind event handlers
- Start Tk event loop

Create GUI Components



Create GUI Components

Frame, Label and Entry



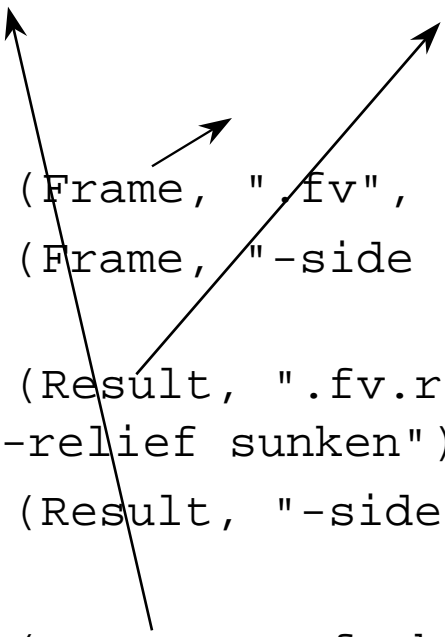
```
Tcl.Tk.Create (Frame, ".amt", "-bd 2");
Tcl.Tk.Pack   (Frame, "-side top -fill x");

Tcl.Tk.Create (Amt_Entry, ".amt.entry",
               "-width 20 -bg white");
Tcl.Tk.Pack   (Amt_Entry, "-side right");

Tcl.Tk.Create (Label, ".amt.label",
               "-text {Monthly Savings Amount:}");
Tcl.Tk.Pack   (Label, "-side right");
```

Create GUI Components

Frame, Button, and Label



The diagram consists of three arrows originating from the right side of the code blocks and pointing towards the top of the slide. The first arrow starts from the 'Frame' argument in the first 'Tcl.Tk.Create' line and points to the 'Frame' component in the title. The second arrow starts from the 'Result' argument in the third 'Tcl.Tk.Create' line and points to the 'Label' component in the title. The third arrow starts from the 'Button' argument in the fifth 'Tcl.Tk.Create' line and points to the 'Button' component in the title.

```
Tcl.Tk.Create (Frame, ".fv", "-bd 2");
Tcl.Tk.Pack   (Frame, "-side top -fill x");

Tcl.Tk.Create (Result, ".fv.result",
               "-width 20 -relief sunken");
Tcl.Tk.Pack   (Result, "-side right");

Tcl.Tk.Create (Button, ".fv.button",
               "-text {Compute Future Value:} " &
               "-command computeFutureValue");
Tcl.Tk.Pack   (Button, "-side right");
```

Create GUI Components

Window Title and Focus

```
-- Add a window title
Tcl.Ada.Tcl_Eval (Interp,
    "wm title . {Future Value of Savings}");

-- Set focus to the first entry field
Tcl.Ada.Tcl_Eval (Interp, "focus
    .amt.entry");
```


Create New Tcl Command

```
declare
    package CreateCommands is new
        Tcl.Ada.Generic_Command (Integer);
    Command : Tcl.Tcl_Command;

begin
    -- Create a new Tcl command to compute
    -- future value.
    Command := CreateCommands.Tcl_CreateCommand (
        Interp, "computeFutureValue",
        Compute_Future_Value_Command'access,
        0, NULL);
```

Create New Tcl Command

Tcl calls Ada Subprogram

```
function Compute_Future_Value_Command (  
  ClientData : in Integer;  
  Interp      : in Tcl.Tcl_Interp;  
  Argc        : in C.Int;  
  Argv        : in CArgv.Chars_Ptr_Ptr  
) return C.Int;  
pragma Convention (C, Compute_Future_Value_Command);  
-- Declare a procedure, suitable for creating a  
-- Tcl command, which will compute the Future Value.
```

Create New Tcl Command

Get the “Monthly Savings Amount”

```
begin -- Compute_Future_Value_Command

    -- get the monthly investment amount from its text
    entry
    -- field, evaluate it in case it is an expression,
    -- and make sure it is not less than zero
    Amount := Money (
        Tcl.Ada.Tcl_ExprDouble (Interp, Tcl.Tk.Get
        (Amt_Entry))) ;
    if Amount < 0.0 then
        return Tcl.TCL_OK ;
    end if ;
```

Create New Tcl Command

Get the “Annual Interest Rate”

```
-- get the annual interest rate from its text entry
-- field, evaluate it in case it is an expression,
-- and make sure it is not less than zero
Annual_Rate := Float (
    Tcl.Ada.Tcl_ExprDouble (
        Interp, Tcl.Tk.Get (Rate_Entry)));
if Annual_Rate < 0.0 then
    return Tcl.TCL_OK;
end if;
```

Create New Tcl Command

Compute and Display Future Value

```
-- compute the monthly interest rate
Rate := Annual_Rate / 1200.0;

-- compute the number of months
Months := Years * 12;

-- compute future value
Future_Value := Money (
    Float (Amount) * ((1.0 + Rate)**Months - 1.0)/Rate);

-- put the future value into the result label
Tcl.Tk.Configure (Result, "-text " &
    Money'image (Future_Value));
```

Bind Event Handlers

```
-- Bind Return to the button command.  
-- If Button has focus, pressing Return will  
-- execute Tcl command, computeFutureValue  
Tcl.Tk.Bind (Button, "<Return>",  
    "computeFutureValue");
```

Start Tcl Interpreter and Initialize Tcl and Tk

```
-- Create Tcl interpreter
Interp := Tcl.Tcl_CreateInterp;
-- Initialize Tcl
if Tcl.Tcl_Init (Interp) = Tcl.TCL_ERROR then
    Text_IO.Put_Line ("FutureValue: Tcl_Init failed: " &
        Tcl.Ada.Tcl_GetResult (Interp));
    return;
end if;
-- Initialize Tk
if Tcl.Tk.Init (Interp) = Tcl.TCL_ERROR then
    Text_IO.Put_Line ("FutureValue: Tcl.Tk.Init failed: " &
        Tcl.Ada.Tcl_GetResult (Interp));
    return;
end if;
```

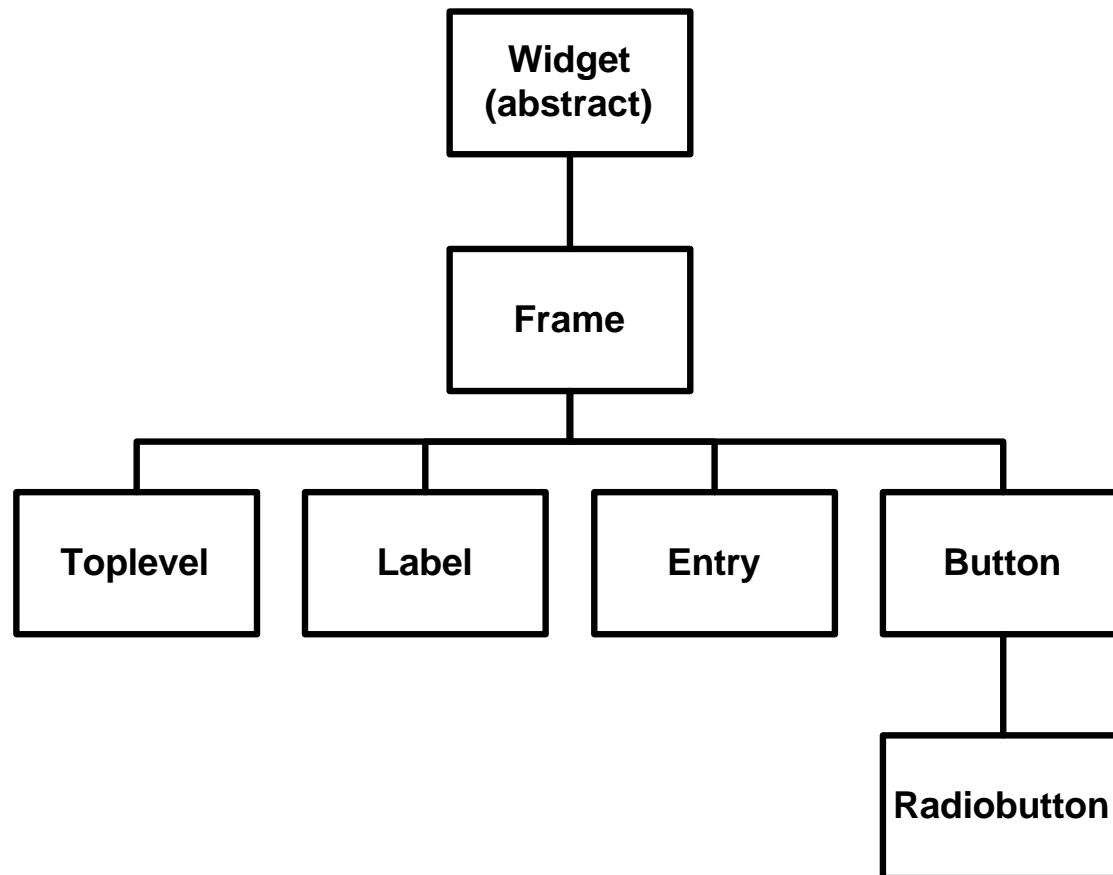
Start Tk Event Loop

```
-- Loop inside Tk, waiting for events to occur and  
-- thus commands to be executed.  
  
-- When there are no windows left or when we execute  
-- "exit" command, Tcl.Tk.MainLoop returns.
```

```
Tcl.Tk.MainLoop;
```


TASH GUI Components

Inheritance Hierarchy



Conclusion: Use TASH

- Use TASH when you want scripting language features in Ada
- Use TASH's Tk interface when
 - Building a small, simple GUI
 - Using TASH already for its scripting language features
- You'll be glad you did!