Wafe –
A Tcl Interface to the X Toolkit
or
A Graphical Frontend for Applications in Various Programming Languages

Version: 0.93

Gustaf Neumann, Stefan Nusser

neumann@wu-wien.ac.at, nusser@wu-wien.ac.at

Wirtschaftsiuniversität Wien
Abteilung für Wirtschaftsinformatik
Augasse 2–6
A-1090 Vienna

June 27, 1993

Abstract

Wafe provides a flexible and easy to use interface to the *X Toolkit* Release 4 or 5 (Xt), the *Athena* widget set (Xaw) and (incomplete) to the OSF/Motif widget set (Xm). Wafe is implemented using the embeddable command language *Tcl*. Wafe allows access to Xt's functionality from all compiler and interpreter languages, provided that they can communicate unbuffered over *stdout* and *stdin*. Furthermore, Wafe can be used with *Tcl* scripts for easily creating new X Window Applications or user interfaces to existing Unix commands.

Contents

1 Introduction

Wafe stands for *widget (Athena) front end* and provides an interface to the X Toolkit. The name Wafe is a historical leftover and not fully appropriate since Wafe supports not only the Athena Widgets but also the Plotter Widget set and the OSF/Motif widget set. In the current release the OSF/Motif support is not complete, interfaces to various routines of the programmatic interface are missing.

Wafe can be used as a standalone prototyping environment for X Toolkit applications or as a frontend to applications programs in various programming languages to provide a graphical user interface without facing the need to enhance the appliction languages with X Window and X Toolkit bindings. The basic idea of the frontend approach is to start an application in two separate processes: one process is dealing purely with application matters, the other one is responsible for the user interface. The two processes communicate using sockets. The application process writes to stdout, its output is parsed and interpreted by the frontend. This way the application process sends send commands or procedures to the fontend process, which builds up the graphical user interface and handles the incoming events.

The purpose of this manual is mostly to provide an overview of the commands available in the Wafe environment. A deeper discussion of the desiogn overview of Wafe can be found in [4]. A more technical reference of the Widget sets and commands can be found in [5, 1, 3, 7]. For more details about the embeddable command languge Tcl refer to [6].

The following section describes how to use Wafe as a frontend and the the command line arguments of Wafe. Section 2 describes the most important Wafe commands (X Toolkit specific commands, Athena Widget specific commands, XPM [2] specific commands). The Short Reference in appendix A on page 40 was generated automatically from the Wafe sources and provides the complete set of Wafe commands and the names of the X Toolkit counterparts.

1.1 Using Wafe as a Frontend

The way Wafe communicates with the application process is controlled by various command line arguments. If none of the options *--d*, *--e*, *--f* or *--v* is specified, the application program is started as a subprocess of *Wafe*. If the *Wafe* application is called *myprog* and a link from *Wafe* to the name *xmyprog* is made, executing *xmyprog* starts *Wafe* with myprog as a subprocess.

```
ln -s /bin/X11/wafe xmyprog
xmyprog
```

If the local Unix variant does not support symbolic links, a hard link may be used instead of the symbolic link in the example above. *Wafe* searches all directories in the path in order to spawn the child-process *myprog* (the name is simply derived by removing the leading character "x"). Alternatively the name of the application program can be specified using the *--p* command line option.

```
wafe --p myprog
```

The application *myprog* looses it's standard input and output streams, diagnostic output will be merged with *Wafe*'s stderr. *Wafe* communicates with the application by reading it's stdout and writing to stdin. Commands to *Wafe* must be prefixed by a so called prompt character, defaulting to "%", otherwise they are just echoed to *Wafe*'s standard output. The prompt character can be changed with the *--c* command line option. Furthermore, if you want *Wafe* to accept commands without prompt character, use the *--n* command line option (This means that the application completely looses its output stream).

The application class is derivated from the link's name by capitalizing the "x" and the first letter of the executable; any dots will be substituted by dashes. This is the name, under which *X* searches the application's defaults file. So, in our example, the class is *XMyprog* and the corresponding app-defaults file will be searched. The application instance name is the link's name with dots substituted by dashes.

Command line arguments can be passed to the client with one restriction: They mustn't begin with the characters "--", since this is reserved for *Wafe*'s arguments. Thus the command line

```
xmyprog filename --n
```

will cause *Wafe* to spawn the process *myprog* with the argument *filename*. The "--n" is one of *Wafe*'s command line arguments, which are described in the next section.

1.2 Command Line Arguments

The following command line options change the behavior of *Wafe*'s communication mechanism:

--i Switches to *input mode*, which means that *Wafe* reads from *stdin* (which it normally doesn't) and forwards the commands to the client-application. Be aware that programs using input-mode can't be executed in the background!

--n Turn of the prompt character mechanism. This causes *Wafe* to regard every message it receives from the application as a *Wafe* command.

--c*c* This option simply allows the user to specify the prompt character which is not used if *--n* is specified. It defaults to "%".

--d Enter *direct mode*; this is a special mode, which I use mostly for debug purposes. In direct mode, you can start *Wafe* itself without a link, just by typing

```
wafe --d --n
```

Commands are now read from *stdin*, output is sent to *stdout*. Additionally, the option *--n* is recommended so that it is unnecessary to prefix every command with the prompt character.

--f Run in *file mode*. The next argument after specifying *--f* must be the name of a file containing valid *Tcl* commands. The file will be executed by the *Tcl source* command. Before executing the file, *Wafe* will set the *Tcl* variables ARGC, ARGV and ENV to the corresponding contents. ARGV and ENV have both *Tcl* list structure.

--p Specify name of client program explicitly. The next argument after *--p* must be the name of the program to be executed as client program. This option can be used alternatively to the standard link mechanism descibed above where the leading "x" is stripped to derive the client program's name.

--v Displays the current version of *Wafe* and exits immediately.

--e Takes the next argument to be a valid Wafe command string and executes it. Note that Wafe doesn't terminate automatically in this mode.

--D*d* Starts Wafe in *debug mode* and passes the specified option to the *debug library*.

2 Wafe Commands

In addition to the standard *Tcl* commands, Wafe makes several new commands available for the client.

Please note that according to the very nature of the *Tcl* language and Wafe's stream oriented communication mechanism, every argument's type has to be *string*. Widgets, for example, are always referenced by their names and conversions are done as necessary.

For the description of the new commands, the following conventions will be kept:

expression$^+$	means one or more occurrences of *expression*
expression*	means zero or more occurrences of *expression*
expression$^?$	means one or zero occurrences of *expression*

To apply these rules on several arguments, angles $\langle\ \rangle$ are used. Terminal symbols are written in sans-serif and nonterminals in *italic* font style. Informal descriptions are typeset in the normal text font style. Commands, which are different from their Xt counterparts are described in detail, those which are added to Wafe without changes only have a short and informal description; it is assumed that you are familiar with these libraries.

2.1 Widget Creating Commands

The widget creating commands all have the same syntax.

widgettype widget parent unmanaged$^?$ $\langle$*attribute value*$\rangle^*$

widgettype	$\longrightarrow$	*shell* | *simple* | *menu* | *text* | *composite* | *constraint*
widget	$\longrightarrow$	string name of instance
parent	$\longrightarrow$	string name of instance | topLevel

shell	$\longrightarrow$	overrideShell | transientShell | topLevelShell
simple	$\longrightarrow$	command | label | grip | repeater | toggle | scrollbar | tree | stripChart | panner | list | mailbox | logo
menu	$\longrightarrow$	simpleMenu | sme | smeBSB | smeLine | menuButton
text	$\longrightarrow$	asciiText
composite	$\longrightarrow$	box | porthole
constraint	$\longrightarrow$	paned | viewport | form | tree | dialog
attribute	$\longrightarrow$	attribute of this widgetclass, please refer to a documentation of the Athena widget set (eg [1] or [7] for a complete list of valid attributes)
value	$\longrightarrow$	type, corresponding to the chosen attribute

The defined constant *topLevel*, is used to reference the application's top level widget, which is created automatically at startup.. For a list of possible attribute/value pairs and detailed information about a widget's functionality compare to [1] or [7]. If desired, a widget can be created without being managed by the parent by stating unmanaged after the parent's name. This can be advantageous if many widgets are created as children from the same parent. They can be managed all together after the creation by using the manageChild command.

If *Wafe* is compiled with the *At Plotter Widget Set*, the following widgets are also available *plotter*, *axis*, *xyLinePlot*, *labelAxis*, *xyAxis*, *linePlot*, *barPlot* and *textPlot*. For their resources and programmatic interface check out the documentation of the plotter widget.

Note that the following command is the only exception from this general structure of the widget creation commands.

applicationShell *widget display* $\langle$*attribute value*$\rangle^*$

display $\longrightarrow$ string name of display

Use this command to create a new application shell widget on the specified display, which will be the top of a new widget tree. Note that this shell widget can be iconified seperately and is subject to the window manager's client decorations.

2.2 Basic Toolkit Commands

This section describes Xt specific commands. Most of these commands correspond to commands described in the *X Toolkit Intrinsics Reference Manual*, where the leading *Xt* or *Xaw* prefix was removed. The *Wafe* developer might wish to consult the **Gen* files in the `src` directory of the *Wafe* distribution which are used as input to the C generator.

setValues *widget* $\langle$*attribute value*$\rangle^+$or

sV *widget* $\langle$*attribute value*$\rangle^+$

> Changes one or more attribute-value pairs of the specified widget. Conversion is done to the corresponding argument type. Note that some attributes, which have no string representations, can't be set by this command (eg the *colormap* attribute). This sort of attributes are either specified in [1] or have to be found out by trial and error. *Wafe* allows the *callback* resource to be set to a *Tcl* command.

value
getValue *widget attribute* or

value
gV *widget attribute*

value	$\longrightarrow$	string, actual data type depending on queried attribute

> The counterpart of *setValues* is *getValue*, which allows a specified attribute to be queried. Note that *Wafe* tries to convert the result to the best possible string representation suitable for Tcl. Some attributes are therefore returned in Tcl list structure (as for example the attributes of type *WidgetList).*

mergeResources *widget* $\langle$*resName value*$\rangle^*$

widget	$\longrightarrow$	widget to determine display
resName	$\longrightarrow$	binding$^?\langle$binding component$\rangle^*$
binding	$\longrightarrow$	"." | "*"

component $\longrightarrow$ "A"-"Z" | "a"-"z" | "0"-"9" | "_" | "-"

This command can be used to write one or more resource specifications into the resource database. It is useful to specify common resources of several widgets at the beginning of the application. Note that with mergeResources you can specify class names as well as instance names since the same syntax as in an application defaults file is used. Therefore your specification can, in contrast to the setValues command, be applied to several widgets in the application.

string
getClass *widget*

This function returns the class of a widget in form of a string

string
getTypeOfAttribute *widget attribute*

Use this function to determine the type of the given attribute in the widgetclass of *widget*. If the widgetclass does not have an attribute with the given name, an empty string is returned.

int
isWidget *widget*

This function returns 0 if no widget with the given name (path) exists currently, otherwise non-zero.

translateCoords *widget x y outX outY*

This command translates *x* and *y* coordinates in the given widget into root window corrdinates. After the execution the *Tcl* variables *outX* and *outY* are bound with the resulting values.

destroyWidget *widget*$^+$

This command takes any number of widget-names as arguments, destroys them completely and releases all allocated resources.

quit

To terminate Wafe use quit. To ensure that the spawned application process will also terminate, Wafe sends it the *SIGTERM* signal. This – of course – won't happen in *file mode* or *direct mode*.

manageChild *widget**

This command takes any number of widgets as arguments and brings them under parental management. This is only necessary, when the widget creating commands are used with the unmanaged argument!

realize

This command realizes all top level application shells. The application's main shell with the name topLevel is created automatically.

unrealizeWidget *widget**

This command takes any number of widgets as arguments and unrealizes them.

popup *popupShell grab*

grab $\longrightarrow$ none | exclusive | nonexclusive

Shell is the name of a previously (either with transientShell or with overrideShell) created popup-shell. The kind of grab used is specified with *grab*, which is the way how user events are constrained to the popup shell or its children. It can be one of the constants none, nonexclusive or exclusive.

popupSpringLoaded *popupShell*

This command displays the specified and previously created *shell* widget as a spring-loaded popup, which means that it is invisible to the window manager and disables user input to all windows except to the popup itself.

popdown *popupShell*

Use popdown to pop down a previously created and mapped popup-shell.

2.3 Callbacks and Actions

The *X-toolkit* provides two mechanisms to link widgets to application code: One of them are callbacks, easy to use for predefined purposes, the other actions, which are more flexible and configurable but involve a more complicated handling. This section describes how these features are implemented in Wafe, if your are not familiar with the basic concepts of callbacks and actions please refer to the Xt Programming Manual [5].

Callbacks which should be executed when certain widget specific conditions occur (such as clicking with pinter button 1 in an Athena command widget) can be specified directly as resources during widget creation or the sV command. The content of the callback reources is a Tcl command which should be executed. The content of a callback resource can be retrieved using gV. When a callback is activated, printf-like % substitutions are performed in a first step. In this way Wafe provides the functionality of *Xt*'s *call Data* mechanism). The table below shows the %-codes for the Athena widget. Please refer to the short reference for a complete summary of the %-codes.

Widgets	*Callback*	*Code*	*Description*	*Type*
all	all	%w	name of widget	string
list	callback	%i	index	int
list	callback	%s	active element	string
scrollbar	scrollProc	%p	position	int
scrollbar	jumpProc	%p	percent	float

Please note that although different types are mentioned in the table, all of these codes will be substituted by a string. The resulting command string will be interpreted using *Tcl* eval. To avoid unexpected side effects due to the substitution, characters like backslash, dollar or square brackets, which have special meanings in Tcl are escaped in %s substitutions using a backslash. %s should be used inside double quoted strings.

callback *widget name type arg**

name	$\longrightarrow$	valid callback attribute of the specified widget
type	$\longrightarrow$	none | exclusive | nonexclusive | popdown | position | positionCursor
arg	$\longrightarrow$	additional arguments depending on type

This command sets the callback resource of the named widget to one of the predefined Toolkit functionc. For valid *types* refer to the following table. Note that the execution of a Wafe command can be registered with the callback resource of a widget with the setValues command.

Type	*Description*
none	realize a popup-shell, grab-type none
exclusive	realize a popup-shell, grab-type exclusive
nonexclusive	realize a popup-shell, grab-type nonexclusive
popdown	unrealize a popup-shell
position	position a popup-shell relative to its parent
positionCursor	position a popup-shell under the cursor

According to the type used, *callback* takes the following additional arguments.

none, exclusive, nonexclusive

callback *widget name* none *popupShell* or

callback *widget name* exclusive *popupShell* or

callback *widget name* nonexclusive *popupShell*

These callbacks all take one additional argument: *shell*, specifying a popup-shell widget which will be realized with grab-type specified as none, exclusive or nonexclusive. Furthermore this callback will set the widget *widget* insensitive.

popdown

callback *widget name* popdown *popupShell enable*

enable $\longrightarrow$ string name of a widget

Takes two additional arguments: *shell* and *name*. It will unrealize *shell* and set the specified widget sensitive.

position

callback *widget name* position *shell basis*

basis $\longrightarrow$ string name of a widget

These callback takes two further arguments: The name of a popup-shell widget and the name of another widget. $W_{a}f^{e}$ attempts to position *shell* relative to the other specified widget *basis*. The full syntax of the last parameter is *widgetname:xoffset/yoffset*. If *yoffset* (and the slash) is omitted, *yoffset* is set to the value of *xoffset*. If *xoffset* is also omitted (no colon) *xoffset* and *yoffset* are set to the value 50.

positionCursor

callback *widget name* positionCursor *shell*

Takes one additional argument, which is the name of a popup-shell widget. It is to be placed at the position of the pointer.

action *widget type translations*

type	$\longrightarrow$	augment | override | replace
translations	$\longrightarrow$	string translation table – refer to the Xt Reference Manual [1], Appendix F, for an exact definition.

This command allows the *translations* resource to be modified. The *type* argument specifies, how the new bindings are merged with the existing ones: override means that a translation specified for an existing event overrides the old action, augment defines precedence for the existing values. The use of replace means the total replacement of the previous bindings by the translation table specified.

If you are using the replace or override directives, you will need information about a widget's actions and default bindings in order to avoid conflicts with your own translations. For a complete description including the new R5 widgets refer to the MIT documentation [7].

In addition to the standard Xt actions, Wafe provides a feature for binding the execution of a Tcl command to an event. The globally registered action

exec(*string**)

takes any number of strings as arguments, which compose together a valid Tcl command and arranges for it to be executed. Note that due to the way Xt handles arguments to actions, you must not use a comma (",") in the command, since it will be regarded as an argument separator and will not be passed to the action!

Assuming an application with an asciiText widget named `text`*, the following statement will arrange for the contents of the Text widget to be sent to the application by specifying a translation for a Key event.*

```
action text override \
      {Meta<Key>s: exec(echo [gV text string])}
```

Whenever the key `s` *is pressed while the Meta-key is held down inside the text widget, the contents of it will be sent to the application.*

One of the big advantages in using actions instead of callbacks is the possibility to access information from the event which triggered the execution. This feature is supported in a restricted fashion by the exec action with printf-like % codes. The valid events and the possible codes are

- ButtonPress, ButtonRelease
- KeyPress, KeyRelease
- EnterNotify, LeaveNotify

Code	*Information*	*Valid Events*
%t	event type	all of the above mentioned events
%w	widget	all of the above mentioned events
%b	number of button	ButtonPress, ButtonRelease
%x	x-coordinate	all of the above mentioned events
%y	y-coordinate	all of the above mentioned events
%X	x-root-coordinate	all of the above mentioned events
%Y	y-root-coordinate	all of the above mentioned events
%a	ascii-character	KeyPress, KeyRelease
%k	keycode	KeyPress, KeyRelease
%s	keysym	KeyPress, KeyRelease

The user has to ensure by a corresponding binding in the translation table that a % substitution ocurrs only with a valid event type. The %t code will return unknown if the event is none of the above mentioned.

> *With the following translation, the keycode, character and keysym will be sent to the application any time a key is pressed in a text widget called* `text`.

```
action text override \
  {<KeyPress>: exec(echo K:%k C:%a KS:%s)}
```

installAccelerators *destination source accelerators*

destination $\longrightarrow$ widget in which events occur
source $\longrightarrow$ widget whose actions are invoked

accelerators	$\longrightarrow$	string accelerator table – refer to the Xt Programming Manual [5].

Accelerators provide a facility by which actions occurring in one widget can be triggered by events in another widget. Two directives controlling the merging process can be specified as a part of the accelerator table: override means that the accelerators have precedence over the source widget's translations, augment means that existing bindings can't be overridden by the accelerators.

callCallbacks *widget callback* ⟨*type data*⟩

callback	$\longrightarrow$	callback resource name
type	$\longrightarrow$	float
data	$\longrightarrow$	string containing a value according to type

Callbacks can be called not only by the widget itself but also by the application using callCallbacks. If it is desired to pass a callData information to the callback procedure, this can be done by specifying type and data, otherwise the callData argument defaults to NULL.

callActionProc *widget eventInfo action arg**

eventInfo	$\longrightarrow$	"" | ⟨buttonEvent *button*⟩ | ⟨keyEvent *key*⟩
button	$\longrightarrow$	button1 | button2 | button3 | button4 | button5
key	$\longrightarrow$	string containing a key
action	$\longrightarrow$	string name of an action procedure
arg	$\longrightarrow$	string

This command can be used to explicitly invoke the action procedure named action of widget with the string arguments arg. The contents of the event union, which is passed to the action procedure, can be influenced with the *eventInfo* argument. If this argument is empty, an *XAnyEvent* structure is passed to the action. If buttonEvent is specified with a certain button, an *XButtonEvent* structure is passed. This structure contains as the *button* member the specified button, all other members are filled automatically. Alternatively, an *XKeyEvent* structure can be passed with the *keycode* member set according to the specified key.

Please note that this command does not offer all the features of it's Xt counterpart, which is due to the very complex structure of the XEvent union.

removeCallbacks *widget callback*

The command removeCallbacks is used to un-register all callback procedures currently associated with the callback resource callback of widget.

setWMProtocols *w*idget protocol

widget	⟶	Name of a widget
protocol	⟶	WM_TAKE_FOCUS | WM_SAVE_YOURSELF | WM_DELETE_WINDOW

Use this command to set the WM_PROTOCOLS property of the specified widget's window (usually a shell widget) to one of the three predefined values.

2.4 Widget specific commands

Besides of the commands which provide the basic functionality, $W_a f^e$ offers widget specific commands – either for reasons of convenience or because some Xt features had to be implemented in a different manner. All of these commands are described in the following sections.

2.4.1 Scrollbar

scrollbarSetThumb *scrollbar top shown*

scrollbar	⟶	name of a scrollbar widget instance
top	⟶	string which contains a float value
shown	⟶	string which contains a float value

This command is not more than a convenience function for accessing the top and shown attributes of a previously created scrollbar widget. It moves the scrollbar's thumb to a new position, where *top* specifies the position of the top and *shown* the length of the thumb, both as a fraction of the scrollbar's total length.

Using this command is saver than using setValues, because setting a float attribute might cause problems on some systems. Note that this command doesn't work if called from a scrollbar's *jumpProc* callback procedure.

2.4.2 StripChart

The stripChart widget provides a real time chart of a single value. It's functionality is made accessible by the following command.

scSet *stripChart value*

stripChart	⟶	name of a stripChart widget instance
value	⟶	string, which contains a double value

This command is used to specify a stripChart widget's current value. The widget will plot a graph of this value by querying it every *update* seconds. The *update* resource can be changed with a call to setValues.

2.4.3 List

listUnhighlight *list*

This command unhighlights the highlighted element of a list widget. Only one element can be highlighted at a time, listChange unhighlights automatically the active element before performing the change.

listHighlight *list index*

index	⟶	string containing an integer value

This command highlights the specified element.

val
listShowCurrent *list*

val	⟶	name of a Tcl array variable, having the elements list_index and string
*val(*list_index*)*	⟶	string containing an integer value
*val(*string*)*	⟶	string value

Use listShowCurrent to get the currently highlighted element and its index. Note that within a callback procedure, you can use the *%i* and *%s* substitution specifiers.

listChange *list nitems longest resize type arg*$^+$

list	⟶	name of a previously created list widget
nitems	⟶	string containing a signed int
longest	⟶	string containing a signed int
resize	⟶	1 | 0
type	⟶	Arg | File
arg	⟶	string depending on type

Use listChange to change the contents of *list*. This command modifies the *list*, *longest* and *nitems* attributes, which mustn't be set with the setValues command! If *nitems* or *longest* contain a value less than one, the list widget will calculate the actual value.

To instantiate a new list widget, first create an empty one with the list widget command, afterwards fill it with listChange. There are two possibilities to specify the contents of *list*, which depend on the value of *arg*.

listChange *list nitems longest resize* File *filename*

filename $\longrightarrow$ valid Unix filename string

In this case the new contents of *list* are read from the file *filename* where each line is taken to be one list element. The carriage-returns and line-feeds are discarded!

listChange *list nitems longest resize* Arg *element*$^+$

element $\longrightarrow$ string

Using the Arg type, the elements of the list widget can be specified as arguments to the listChange command. Each argument is regarded as one list element.

If the execution of a *Tcl* command is registered with the callback attribute called *callback*, the current element and it's index can be accessed with the %i and %s substitution codes (see also callback).

The contents of a list widget can also be changed via *setValues*. Be sure that at any time the *numberStrings* resource is set to a value equal or lower to the number of strings in the list. Otherwise the list widget produces an unfriendly core dump.

listAppend *list element*$^+$

This command can be used to append one or more elements to an existing list. It is no generic Athena command; the array of strings is merely rebuild with the new values appended and afterwards the value of the *list* resource is updated.

The following program creates a list widget, reads in the contents of the current directory and sends the name of the chosen file to the application.

```
#!/usr/bin/X11/wafe --f
list browse topLevel callback {echo You selected: %s}
eval [concat listChange browse 0 0 1 Arg \
         [split [exec /bin/ls -1] \n]]
realize
```

Assuming the user selects the file `myprog` *with the pointer, the application will receive* "`You selected:  myprog`" *on* stdin. *Note that* `eval`, `concat`, `split` *and* `exec` *are standard Tcl commands. For further information refer to the Tcl manual [6].*

2.4.4 AsciiText

One of the most important widgets is the text widget. $W_a f^e$ provides only the Athena asciiText widget, which should be quite enough for most of the applications. Note that a great part of asciiText's functionality can be accessed via actions – if you want them to be triggered by the application rather then by the user, use callActionProc. Note that not all of the functions begin with "asciiText", since Some of them are made availaible by asciiText's subparts.

asciiTextAppend *asciiText string*

> *asciiText* ⟶ name of a previously created asciiText widget
>
> This command can be used to append *string* to the contents of *asciiText*, presuming that the asciiText widget is not displaying a file (which means in other words that the attribute *type* must not have the value *File*). Note that this is no original Athena command and simply reads the value of the text widget's string resource, generates a new one by appending the specified argument and finally performs a SetValues to set the new resource value.

asciiTextSaveAsFile *asciiText filename*

> This command saves the contents of *asciiText* to the specified file, regardless wether a file or a string is edited.

result
asciiSave *asciiText*

> *result* ⟶ 1 | 0
>
> This command writes out the contents of an asciiText widget of type *file*, given that you made any changes since the last save.

result
asciiSourceChanged *asciiText*

result	⟶	1 | 0

Use this function to determine wether the source has changed since the last save.

textSinkSetTabs *asciiText tab**

tab	⟶	string which contains an integer

Use this command to specify the distances between *asciiText*'s tab-stops. The unit which is used to measure the distance is the width of a figure in the used font.

textSetSelectionArray *asciiText selectType** selectNull

selectType	⟶	selectAll | selectChar | selectLine | selectParagraph | selectPosition | selectWord

This command is used to configure the asciiText widget's behavior for multi click text-selections. Each *selectType* argument corresponds to one click, the last argument must be selectNull. If an internal timeout value is exceeded, the widget restarts counting.

textPosition
textSearch *asciiText scanDir txt*

textPosition	⟶	string containing a long integer value
scanDir	⟶	left | right
txt	⟶	name of a *Tcl* array variable having the members firstPos, ptr and length
*txt(*firstPos*)*	⟶	string containing an integer value
*txt(*length*)*	⟶	string containing an integer value
*txt(*ptr*)*	⟶	a string

Use textSearch to search in the text displayed by the text widget for the beginning of the specified textBlock. This command returns a value less than 0 upon unsuccesfull termination, otherwise the position of the beginning of the searched block.

> *The following commands search an asciiText widget named* text *for the word* example *and move the insertion Point to this location.*

```
set textBlock(ptr) "example"
set textBlock(length) 7
set textBlock(firstPos) 0
textSetInsertPosition text\
      [textSearch text right textBlock]
```

In most cases, when a C-function takes a structure as an argument, we implemented the corresponding W_af^e command in this style. Usually the members of the *Tcl* array have the same names as the C-language structure members.

read
textSourceRead *asciiText begin txt length*

read	⟶	string containing a long integer value
begin	⟶	string containing a long integer value
length	⟶	string containing an integer value
txt	⟶	name of a Tcl variable to be set as result; it will have the following members: firstPos, length and ptr
*txt(*firstPos*)*	⟶	string containing an integer value
*txt(*length*)*	⟶	string containing an integer value
*txt(*ptr*)*	⟶	a string

Use textSourceRead to read the text of the specified widget. It may takle several calls to this function to read the requested amount of text; the number of characters actually read is returned and can be compared to the requested number.

found
textSourceScan *asciiText begin boundary direction count include*

found	⟶	string containing a long integer
begin	⟶	string containing a long integer
boundary	⟶	position | whiteSpace | EOL | paragraph | all
direction	⟶	left | right
count	⟶	string containing an integer
include	⟶	true | false

Use this command to search the widget from *begin* on in the specified direction for one of the predefined boundaries.

textInvalidate *asciiText begin end*

begin	⟶	string containing a long integer value
begin	⟶	string containing a long integer value

This function causes the specified range from *begin* to *end* to be redisplayed.

lines
textSinkMaxLines *asciiText height*

lines	⟶	string containing an integer value
height	⟶	string containing a long integer value

This command returns the maximum number of lines that will fit in a window with the specified height.

height
textSinkMaxHeight *asciiText lines*

height	⟶	string containing an integer value
lines	⟶	string containing an integer value

Calculates the height that will taken by the specified number of lines.

textEnableRedisplay *asciiText*

This function flushes any changes due to patched updates.

textDisableRedisplay *asciiText*

To disable redisplay while making several changes use textDisable-Redisplay.

textDisplay *asciiText*

This function forces redisplay of all accumulated updates.

topPosition
textTopPosition *asciiText*

topPosition $\longrightarrow$ a string containing a long integer value

Obtain the character position of the left-most character of the first line displayed.

textSetInsertionPoint *asciiText insertPosition*

insertPosition $\longrightarrow$ string containing a long integer value

Use this convenience function to set the value of the insertPosition resource.

insertPosition
textGetInsertionPoint *asciiText*

Use this convenience function to get the value of the insertPosition resource.

textDisplayCaret *asciiText trueFalse*

trueFalse	$\longrightarrow$	true | false

This convenience function can be used to modify the value of the displayCaret resource.

textGetSelectionPosition *asciiText beginVar endVar*

beginVar	$\longrightarrow$	name of a Tcl variable
endVar	$\longrightarrow$	name of a Tcl variable

Use this function to retrieve the text, which is currently selected. It will set the two *Tcl* variables as specified iby the arguments. If the values are equal, nothing is currently selected!

textSetSelection *asciiText begin end*

begin	$\longrightarrow$	string containing a long integer value
end	$\longrightarrow$	string containing a long integer value

This function sets the PRIMARY selection buffer fom character position *begin* to position *end* and highlights this area.

textUnsetSelection *asciiText*

To unhighlight a previously highlighted text use textUnsetSelection.

result
textReplace *asciiText begin end txt*

result	$\longrightarrow$	editDone | positionError | editError
begin	$\longrightarrow$	string containing a long integer value
end	$\longrightarrow$	string containing a long integer value
txt	$\longrightarrow$	name of a *Tcl* array having the members firstPos, ptr, length

*txt(*firstPos*)*	⟶	string containing an integer value
*txt(*length*)*	⟶	string containing an integer value
*txt(*ptr*)*	⟶	a string

Use this function to replace the text between *start* and *end*.

asciiSourceFreeString *asciiText*

This function frees the memory which is allocated after a GetValues command in order to build up the string. Note that this memory is automatically freed with the next getValues call!

2.4.5 Tree

treeForceLayout *tree*

tree	⟶	name of previously created tree widget instance

This command will force the tree widget to recalculate its layout and to arrange its children. If the *autoReconfigure* resource is set to false, this provides an efficient way to layout the tree.

2.4.6 Form

formDoLayout *form layout*

form	⟶	name of previously created form widget instance
layout	⟶	1 | 0

When making several changes to the children of a form, it might be desirable to set *layout* to false until after all changes have been made.

2.4.7 Paned

panedAllowResize *child allow*

paned	⟶	name of a Paned widget's child
allow	⟶	1 | 0

Equivalent to changing the *allowResize* constraint resource of the specified child.

panedSetMinMax *child min max*

min	⟶	string containing an integer
max	⟶	string containing an integer

Equivalent to setting the *min* and *max* constraint resources of the specified child.

panedGetMinMax *child minVar maxVar*

minVar	⟶	name of a *Tcl* variable
maxVar	⟶	name of a *Tcl* variable

Equivalent to querying the *min* and *max* constraint resources of the child. The specified *Tcl* variables will be set by the command.

panedSetRefigureMode *child mode*

paned	⟶	name of a child widget of a Paned widget
mode	⟶	1 | 0

Use this command to enable and disable automatic recalculation of pane sizes and positions.

num
panedGetNumSub *paned*

num	⟶	string containing an integer value
paned	⟶	name of a previously created paned widget

This functions returns the number of children. Note that the *grip* widgets are children of the Paned widget as well!

2.4.8 Toggle

toggleChangeRadioGroup *toggle rGroup*

toggle	⟶	name of a previously created toggle widget
rGroup	⟶	name of a previously created toggle widget | NULL.

Use this command to add *toggle* to an existing radio group (*rGroup* can be any member widget of this group) or to remove a toggle from a radio group (by specifying NULL as *rGroup*.

toggleUnsetCurrent *rGroup*

Use this command to unset all toggles in the specified radio group.

2.4.9 SimpleMenu

entry
simpleMenuGetActiveEntry *simpleMenu*

entry	⟶	name of an Object
simpleMenu	⟶	name of a previously created simpleMenu widget

This function returns the name of the currently avtive Object in the specified SimpleMenu widget simpleMenuClearActiveEntry *simpleMenu*

This function unhighlights the currently selected entry.

2.5 Communication and Handlers

This section explains, how Wafe's communication mechanism is used to transfer commands and data between Wafe and the application and how it can be modified by handlers. Note that an essential part of this topic is the use of the different command line arguments, which were already described earlier.

2.5.1 The echo command

echo *string*$^+$

string $\longrightarrow$ character string

Communication with the application is primarily facilitated by the command *echo*. It takes any number of strings as arguments, which are send to the application. In *direct mode* or *file mode*, *echo* will behave just like an ordinary print command, writing its arguments on standard output. (Refer eg to the action command in order to see an example).

2.5.2 Handlers

Handlers offer the possibility to change Wafe's communication mechanism. Normally, three streams are assigned to each application, which are redirected as described in the introduction. The *stdout* stream is echoed to the user, after being searched for commands (beginning with the *prompt-character*), *stderr* is merged with Wafe's *stderr* and afterwards echoed to the user and the application's *stdin* is normally used to receive the front end's inputs. An application running in *input mode* will furthermore receive the typed input from the associated terminal.

This standard configuration can be changed by command line arguments and by using handlers. The latter provide a mean to register a certain command with *Wafe*, which is to be executed whenever input is pending on the specified stream. The use of handlers is facilitated by the following commands.

register *source* command

source	$\longrightarrow$	stdin | stdout | stderr | xioerr
command	$\longrightarrow$	valid *Wafe*command string

Use this command to install handler procedures, which are invoked under different conditions, as specified by the argument *source*.

By specifying stdin, stdout or stderr as first argument, *command* will be executed whenever data is pending on the corresponding stream. Before executing the command, *Wafe* will update a global Tcl variable called stdin, stdout or stderr according to the handler registered, which will contain the pending data.

Note that the registration of a handler procedure can change *Wafe*'s standard behaviour: Registering an error-handler means that no error messages of the application will be echoed to the user's terminal. This type of handler can not be used in *direct or* file mode. Output-handlers can not be used in *direct mode – if it is done anyway* Wafe *writes an error message and ignores the command. Registering an output handler obviously makes only sense if* Wafe *is running in* prompt mode. Finally, registering an input handler is not possible in *direct mode and makes only in* input mode sense.

Wafe allows XIO errors to be caught by using the constant xioerr as first argument to the register command. Note that this is a fatal error condition and that *Wafe* will anyway try to kill the application program before exiting afterwards. This type of handler procedure facilitates therfore cleanup operations.

unregister *source*

source	$\longrightarrow$	stdin | stdout | stderr | xioerr

This command is used to unregister the corresponding handler, which means that the default handler will be reinstalled.

Note that a handler procedure is called several times if the message consists of more than one line (Lines are separated by carriage returns). If more than one handler for the same input source is registered, the most recently created one will be used. Unregistring a default handler doesn't work and will lead to a warning message! Let the whole thing be illustrated by the following example.

> *Assuming that the application consists of an asciiText widget named* `text`*, the following code will popup a warning shell, whenever an error occurs in the application.*

```
transientShell errorShell text
command error errorShell
callback errorShell popupCallback positionCursor
callback message callback popdown errorShell output

register STDERR \
    {sV error label $STDERR; popup errorShell exclusive}
```

> *Whenever the application writes something on stderr, the popupshell will pop up – positioned under the cursor – and display the error. Note that the error is displayed in a command widget, which means that the shell can be dismissed by button-click.*

2.5.3 The additional communication channel

Wafe provides another facility which is used mostly to transfer large amounts of data from the application to the front end (eg the contents of an asciiText widget). When the application process is started, *Wafe* creates an additional socket-based communication channel. The following commands provide access to this feature.

fd
getChannel

fd ⟶ string containing an integer value

This command returns the file descriptor to be used by the application, which is under UNIX a small integer referring to an open file.

Note that this way of communication can only be used if the application is able to write directly to a file descriptor. This way how to achieve this depends on the particular interpreter or compiler of the application programming language. So before the additional communication channel is used, the file descriptor should be queried and the information be sent to the application.

setCommunicationVariable *varName length command*

varName	$\longrightarrow$	Valid Tcl variable name
length	$\longrightarrow$	string containing an integer value
command	$\longrightarrow$	valid Tcl command string

Use setCommunicationVariable to make the transferred data availaible for the user interface. As soon as the data with *length* number of bytes is received, Wafe will create the Tcl variable *varName*, which will contain the transferred data, and execute *command*. If the variable allready exists, the previous contents will be deleted. Note that this command does not depend on the data being allready availaible – if the transfer is complete, *command* will be executed immediately, if *length* bytes of data are not yet received, the execution is delayed until the transfer is complete.

2.5.4 Communicating from an Application Program to Wafe

If Wafe is used as a frontend, lines which are starting with the prompt character (usually "%") printed to stdout from an application program are interpeted as Wafe (Tcl) commands. Therefore, the remainder of these lines must be valid Tcl statements, using the appropriate quoting and escaping of literals.

In certain cases, the interpretation of the commands is undesired. Uninterpreted data transfer can be achieved using the communication channel explained above. However, for certain interpreted programming languages, writing to the communication channel hard to achieve, or complicated. Wafe provides starting with version 0.95 means for uninterpreted assignment from client applications to Tcl variables.

If the application program running under Wafe writes

```
%=myvar This will become the content of the TCL-Variable myvar
```

to stdout, the Tcl variable `myvar` is set to the string starting with 'This' and ending with 'myvar'.

The uninterpeted assignment works as follows:

- If the first character after '%' (or the character provided via the --c option) is '=', the next space terminated token is used as a variable name, and the remainder of the line is assigned to the named Tcl variable.
- If the character after the '%' is a '+' followed by a space terminated token for the variable name then remainder is appended to that variable.
- If the character after the '%' is a '\' followed by a space terminated token for the variable name then remainder and a newline character are appended to that variable.

Using this mechanism, "virtual files" can be implemented, which write into Tcl variables. An open command clears the specified variable, a write command appends to it, a newline command appends a newline and the close command just flushes stdout.

2.6 The selection mechanism

X defines an inter-program communication protocol (called ICCCM), which allows X-window based applications to exchange data. This facility is best known when text data is exchanged with the cut-and-paste mechanism. *Wafe* provides simplified access to the exchange of text selections with the following three commands.

Boolean
ownSelection *selection* ⟨NULL | *lostCom*⟩ ⟨NULL | *doneCom*⟩

selection	⟶	character string
lostCom	⟶	valid *Wafe* command string
doneCom	⟶	valid *Wafe* command string

Use the command ownSelection to claim ownership of the *PRIMARY* selection. The specified string *selection* will be placed in the buffer and be available for other applications. This means it can be "pasted" in a window of an application supporting this communication protocol.

This command takes two additional arguments, both of them valid $W_a f^e$-commands. By specifying *lostCom*, a command can be registered, which will be executed when the application looses ownership of the *PRIMARY* buffer. If *doneCom* is specified, the registered command will be evaluated whenever the selection transfer process is complete (which normally means a paste action). Specify the constant NULL if you do not want to use this feature.

getSelectionValue *selHandler*

selHandler $\longrightarrow$ valid $W_a f^e$ command string

This command is the counterpart of ownSelection, it is used to retrieve the current contents of the *PRIMARY* buffer. The registered command string will be ececuted as soon as the selection data is availaible. The global *Tcl* variable PRIMARY will be set to pass the requested information. This command can only be used if the application is realized!

disownSelection

This

command can be used if the application explicitly wants to disclaim its ownership of the *PRIMARY* buffer. Note that the application automatically looses the ownership, whenever an other application claims to be the owner.

A command registered with getSelectionValue will be invoked with an empty string if no data is currently selected. Execution will not be postponed until selection data is availaible! The selection mechanism shall be illustrated by the following example.

The following Wafe-script displays the current value of the primary selection buffer in a label and allows the user to

- *set the selection buffer (Btn1) to the value of the label (eg. the user has slected in the meantime another content, and wants to set the selection buffer to the old value of the label),*
- *read again the selection buffer, set the label to the new value (Btn2) and to highlight the widget, while the button is pressed (this is achieved by the command widget action* `set()`*, or to*
- *quit the program (Btn3).*

```
#!/usr/bin/X11/wafe --f
sV topLevel allowShellResize true
command sel topLevel

action sel override  {<Btn1Down>: \
     exec(ownSelection [gV sel label]) set()}
action sel override  {<Btn2Down>: \
     exec(getSelectionValue {sV sel label $PRIMARY})}
action sel override  {<Btn3Down>: exec(quit)}

realize
getSelectionValue {sV sel label $PRIMARY}
```

At startup, the program shows whatever is selected at this moment. The user can make this text again the PRIMARY selection just by pressing the left pointer button within the window. The application is terminated by pressing the right button.

Beside of those commands, which are the counterparts of the highlevel Xt functions, Wafe offers also some of the lower level Xlib functions. They can be used to modify the *primary* selection buffer as well as the seven *cut buffers*.

convertSelection *widget targetBuffer*

widget ⟶ widget to determine display
targetBuffer ⟶ 0 | 1 | 2 | 3 | 4 | 5 | 6

Use convertSelection to request the current value of the *primary* selection and place it in cut buffer *targetBuffer* for further use.

selection
fetchBuffer *widget cutBuffer*

selection	⟶	string value
widget	⟶	widget to determine display
cutBuffer	⟶	0 | 1 | 2 | 3 | 4 | 5 | 6

Use this function to retrieve the value of the specified cut buffer.

selection
fetchBytes *widget*

selection	⟶	string value
widget	⟶	widget to determine display

Convenience function, which calls convertSelection and fetchBuffer. The value of the *primary* selection is returned. Note that the contents of cut-buffer-0 will get lost!

storeBuffer *widget value cutBuffer*

widget	⟶	widget to determine display
value	⟶	string value
cutBuffer	⟶	0 | 1 | 2 | 3 | 4 | 5 | 6

Use storeBuffer to make *value* the content of the specified cut buffer.

2.7 Integration of the XPM image format

Wafe offers two commands, which provide – together with Arnaud Le Hors *xpm* library – a mean to include multicolored images in your application (For details please refer to the xpm documentation [2]).

First of all, a new string-to-pixmap converter was registered. Note that most of the attributes which currently use bitmaps (eg the resource *bitmap* of the label widget) are actually of type *Pixmap*. If a bitmap is specified, it is rendered using the *foreground* and *background* colors, which are specified as seperate resources. The new converter procedure first attempts to convert the specified filename into a bitmap, using the standard Xt algorithm. If this conversion fails, it tries to interprete the file as an *xpm* image.

Assuming that wafe.xpm is a file in valid xpm format located in the current directory, the following Wafe-script will generate a label showing the image.

```
#!/usr/bin/X11/wafe --f
label image topLevel bitmap wafe.xpm
realize
```

Please note that this conversion allocates color cells, which won't be freed until the application terminates. The allocated resources are managed by Xt's caching mechanism, which means that a pixmap which is converted once, will be availaible for the application's lifetime. Furthermore, if the pixmap file is not located in the current directory, the filename has to be specified with an absolute path.

changePixmap *widget attribute filename*

attribute	⟶	an attribute of type *Pixmap*
filename	⟶	UNIX filename string | None

Use changePixmap to set a resource of type *Pixmap* to the specified image. Obviously, you could achieve the same result by using the followig command

setValues *widget attribute filename*

The only difference is that changePixmap keeps track of all converted pixmaps and – as soon as the attribute is set to a new value – frees all resources (allocated memory as well as color cells). So it is up to you wether you want the benefits of Xt's caching mechanism or wether you prefer having the color cells again availaible.

If you specify None as the pixmap's filename, the current pixmap of *widget* will be destroyed and all allocated resources freed.

setIconPixmap *widget* filename

widget	⟶	name of a shell widget
filename	⟶	UNIX filename string

This command can be used to generate a multicolor XPM-icon for the specified shell widget. If clip-mask information is available it will be used correctly.

A Short Reference

A.1 Motif Widgets and Commands

w **mArrowButton** .... *XmArrowButton*

c **activateCallback**	%c	click count

w **mArrowButtonGadget** .... *XmArrowButtonGadget*

c **activateCallback**	%c	click count

w **mBulletinBoard** .... *XmBulletinBoard*

w **mBulletinBoardDialog** .... *XmCreateBulletinBoardDialog*
for callbacks and functions see: *mDialogShell*

w **mCascadeButton** .... *XmCascadeButton*

w **mCascadeButtonGadget** .... *XmCascadeButtonGadget*

f **mCascadeButtonGadgetHighlight** *XmCascadeButtonGadgetHighlight*

Arguments: cascadeButtonGadget to be highlighted, highlight (bool)
Return value: void

f **mCascadeButtonHighlight** .... *XmCascadeButtonHighlight*

Arguments: widget to be highlighted, highlight (bool)
Return value: void

w **mCommand** .... *XmCommand*

c **commandChangedCallback**		
commandEnteredCallback	%s	command string

f **mCommandAppendValue** .... *XmCommandAppendValue*

Arguments: command widget, command string
Return value: void

f **mCommandError** .... *XmCommandError*

Arguments: command widget, error message
Return value: void

w **mDialogShell** .... *XmDialogShell* *w* **mDrawingArea** .... *XmDrawingArea*

c **exposeCallback**
inputCallback
resizeCallback %W window

w **mDrawnButton** .... *XmDrawnButton*

c **activateCallback** %c click count

c **exposeCallback**
resizeCallback %w window

w **mFileSelectionBox** .... *XmFileSelectionBox*

c **okCallback**
noMatchCallback
applyCallback %d base directory

c **okCallback**
noMatchCallback
applyCallback %p search pattern

c **okCallback**
noMatchCallback
applyCallback %m value of dirMask

c **okCallback**
noMatchCallback
applyCallback %s value of dirSpec

f **mFileSelectionBoxGetChild** .... *XmFileSelectionBoxGetChild*

Arguments: message box widget, child reference
Return value: Widget

w **mFileSelectionDialog** .... *XmCreateFileSelectionDialog*
for callbacks and functions see: *mDialogShell*

f **mFileSelectionDoSearch** .... *XmFileSelectionDoSearch*

Arguments: file selection box, directory mask
Return value: void

w **mForm** .... *XmForm*

w **mFormDialog** .... *XmCreateFormDialog*
for callbacks and functions see: *mDialogShell*

w **mFrame** .... *XmFrame*

w **mInformationDialog** .... *XmCreateInformationDialog*
for callbacks and functions see: *mDialogShell*

w **mLabel** .... *XmLabel*

w **mLabelGadget** .... *XmLabelGadget*

w **mList** .... *XmList*

c **browseSelectionCallback** **defaultActionCallback** **singleSelectionCallback**	%s	selected item
c **browseSelectionCallback** **defaultActionCallback** **singleSelectionCallback**	%p	item position
c **extendedSelectionCallback** **multipleSelectionCallback**	%S	selected items
c **extendedSelectionCallback** **multipleSelectionCallback**	%P	selected item positions
c **browseSelectionCallback** **defaultActionCallback** **extendedSelectionCallback** **multipleSelectionCallback** **singleSelectionCallback**	%t	selection type

w **mMainWindow** .... *XmMainWindow*

f **mMainWindowSetAreas** .... *XmMainWindowSetAreas*

Arguments: main window widget, menu bar widget, command window widget, horizontal scrollbar widget, vertical scrollbar widget, work region widget
Return value: void

w **mMenuBar** .... *XmCreateMenuBar*
for callbacks and functions see: *mRowColumn*

w **mMenuShell** . . . . *XmMenuShell*

w **mMessageBox** . . . . *XmMessageBox*

f **mMessageBoxGetChild** . . . . *XmMessageBoxGetChild*

Arguments: message box widget, child reference
Return value: Widget

w **mMessageDialog** . . . . *XmCreateMessageDialog*
for callbacks and functions see: *mMessageBox*

w **mOptionMenu** . . . . *XmCreateOptionMenu*
for callbacks and functions see: *mRowColumn*

w **mPanedWindow** . . . . *XmPanedWindow*

w **mPopupMenu** . . . . *XmCreatePopupMenu*
for callbacks and functions see: *mRowColumn*

w **mPromptDialog** . . . . *XmCreatePromptDialog*
for callbacks and functions see: *mDialogShell*

w **mPulldownMenu** . . . . *XmCreatePulldownMenu*
for callbacks and functions see: *mRowColumn*

w **mPushButton** . . . . *XmPushButton*

c **activateCallback**	%c	click count

w **mPushButtonGadget** . . . . *XmPushButtonGadget*

c **activateCallback**	%c	click count

w **mQuestionDialog** . . . . *XmCreateQuestionDialog*
for callbacks and functions see: *mDialogShell*

w **mRadioBox** . . . . *XmCreateRadioBox*
for callbacks and functions see: *mRowColumn*

w **mRowColumn** . . . . *XmRowColumn*

w **mScale** . . . . *XmScale*

c **dragCallback**		
valueChangedCallback	%v	value

w **mScrollBar** .... *XmScrollBar*

c **toBottomCallback**
toTopCallback %p pixel

c **decrementCallback**
dragCallback
incrementCallback
pageDecrementCallback
pageIncrementCallback
toBottomCallback
toTopCallback
valueChangedCallback %v value

w **mScrolledList** .... *XmCreateScrolledList*
for callbacks and functions see: *mList*

w **mScrolledText** .... *XmCreateScrolledText*
for callbacks and functions see: *mText*

w **mScrolledWindow** .... *XmScrolledWindow*

w **mSelectionBox** .... *XmSelectionBox*

c **applyCallback**
noMatchCallback
okCallback %v value

w **mSelectionDialog** .... *XmCreateSelectionDialog*
for callbacks and functions see: *mDialogShell*

w **mSeparator** .... *XmSeparator*

w **mSeparatorGadget** .... *XmSeparatorGadget*

w **mSimpleCheckBox** .... *XmCreateSimpleCheckBox*
for callbacks and functions see: *mRowColumn*

w **mSimpleMenuBar** .... *XmCreateSimpleMenuBar*
for callbacks and functions see: *mRowColumn*

w **mSimpleOptionMenu** .... *XmCreateSimpleOptionMenu*
for callbacks and functions see: *mRowColumn*

w **mSimplePopupMenu** .... *XmCreateSimplePopupMenu*
for callbacks and functions see: *mRowColumn*

w **mSimplePulldownMenu** .... *XmCreateSimplePulldownMenu*
for callbacks and functions see: *mRowColumn*

w **mSimpleRadioBox** .... *XmCreateSimpleRadioBox*
for callbacks and functions see: *mRowColumn*

w **mText** .... *XmText*

c **losingFocusCallback**
modifyVerifyCallback
motionVerifyCallback %c position insert cursor

c **losingFocusCallback**
modifyVerifyCallback
motionVerifyCallback %n new position of insert cursor

c **losingFocusCallback**
modifyVerifyCallback %s startPos

c **losingFocusCallback**
modifyVerifyCallback %e endPos

c **modifyVerifyCallback** %t text

w **mTextField** .... *XmTextField*

w **mToggleButton** .... *XmToggleButton*

c **valueChangedCallback** %s state

f **mToggleButtonGetState** .... *XmToggleButtonGetState*

Arguments: toggle button widget
Return value: Boolean

f **mToggleButtonSetState** .... *XmToggleButtonSetState*

Arguments: toggle button widget, state (bool), notify (bool)
Return value: void

w **mToggleButtonGadget** .... *XmToggleButtonGadget*

c **valueChangedCallback** %s state

f **mToggleButtonGadgetGetState** .. *XmToggleButtonGadgetGetState*

Arguments: toggle button widget
Return value: Boolean

f **mToggleButtonGadgetSetState** ... *XmToggleButtonGadgetSetState*

Arguments: toggle button gadget, state (bool), notify (bool)
Return value: void

f **mAddTabGroup** .... *XmAddTabGroup*

Arguments: widget to be added
Return value: void

w **mWarningDialog** .... *XmCreateWarningDialog*
for callbacks and functions see: *mDialogShell*

w **mWorkingDialog** .... *XmCreateWorkingDialog*
for callbacks and functions see: *mDialogShell*

A.2 XmGraph Extension to Motif Widgets

w **mArc** .... *XmArc*

c **arcEditedCallback**	%f	old from
c **arcEditedCallback**	%t	old to
c **arcEditedCallback**	%F	new from
c **arcEditedCallback**	%T	new to

f **mArcGetPos** .... *XmArcGetPos*

Arguments: name of arc Widget, out: x1, out: y1, out: x2, out: y2
Return value: void

w **mGraph** .... *XmGraph*

c **newArcCallback**
newNodeCallback
arcMovedCallback
defaultActionCallback
selectNodeCallback
selectArcCallback

deselectCallback
selectSubgraphCallback %S selected Widgets

c **newArcCallback**
newNodeCallback
arcMovedCallback
defaultActionCallback
selectNodeCallback
selectArcCallback
deselectCallback
selectSubgraphCallback %S selected Arcs

c **arcMovedCallback** %f old from

c **arcMovedCallback** %t old to

c **arcMovedCallback** %F new from

c **arcMovedCallback** %T new to

w **mScrolledGraph** .... *XmCreateScrolledGraph*
for callbacks and functions see: *mGraph*

A.3 Athena Widgets and Commands

w **asciiText** .... *asciiText*

f **asciiSave** .... *XawAsciiSave*

Arguments: ascii text wiget
Return value: Boolean

f **asciiSaveAsFile** .... *XawAsciiSaveAsFile*

Arguments: ascii text wiget, file name
Return value: int

f **asciiSourceChanged** .... *XawAsciiSourceChanged*

Arguments: ascii text wiget
Return value: Boolean

f **asciiSourceFreeString** .... *XawAsciiSourceFreeString*

Arguments: ascii text wiget

Return value: void

f **asciiTextAppend** . . . . *asciiTextAppend*

Arguments: ascii text widget, string to append
Return value: void

w **box** . . . . *box*

w **clock** . . . . *clock*

w **command** . . . . *command*

w **dialog** . . . . *dialog*

f **dialogGetValueString** . . . . *XawDialogGetValueString*

Arguments: dialog widget
Return value: String

w **form** . . . . *form*

f **formDoLayout** . . . . *XawFormDoLayout*

Arguments: form widget, boolean
Return value: void

w **grip** . . . . *grip*

w **label** . . . . *label*

w **list** . . . . *list*

c **callback**	%i	index
c **callback**	%s	selection

f **listAppend** . . . . *listAppend*

Arguments: list widget, items
Return value: void

f **listChange** . . . . *listChange*

Arguments: list widget, number of items, longest, resize, Arg items | File filename
Return value: void

f **listHighlight** .... *XawListHighlight*

Arguments: list widget, item number
Return value: void

f **listShowCurrent** .... *XawListShowCurrent*

Arguments: list widget
Return value: XawListReturnStruct *

f **listUnhighlight** .... *XawListUnhighlight*

Arguments: list widget
Return value: void

w **logo** .... *logo*

w **mailbox** .... *mailbox*

w **menuButton** .... *menuButton*

w **paned** .... *paned*

f **panedAllowResize** .... *XawPanedAllowResize*

Arguments: paned widget, boolean
Return value: void

f **panedGetMinMax** .... *XawPanedGetMinMax*

Arguments: paned widget, min, max
Return value: void

f **panedGetNumSub** .... *XawPanedGetNumSub*

Arguments: paned widget
Return value: int

f **panedSetMinMax** .... *XawPanedSetMinMax*

Arguments: paned widget, min, max
Return value: void

f **panedSetRefigureMode** .... *XawPanedSetRefigureMode*

Arguments: paned widget, boolean
Return value: void

w **scrollbar** . . . . *scrollbar*

c **scrollProc**	%p	position of thumb (in pixel)
c **jumpProc**	%p	floating position (0.0-1.0)

f **scrollbarSetThumb** . . . . *XawScrollbarSetThumb*

Arguments: scrollbar widget, top, shown
Return value: void

f **scSet**
alias; for arguments see: *stripChartSet*

w **simple** . . . . *simple*

w **simpleMenu** . . . . *simpleMenu*

f **simpleMenuClearActiveEntry** . *XawSimpleMenuClearActiveEntry*

Arguments: widget
Return value: void

f **simpleMenuGetActiveEntry** . . . . . *XawSimpleMenuGetActiveEntry*

Arguments: widget
Return value: Widget

w **sme** . . . . *sme*

w **smeBSB** . . . . *smeBSB*

w **smeLine** . . . . *smeLine*

w **stripChart** . . . . *stripChart*

f **stripChartSet** . . . . *stripChartSet*

Arguments: strip chart widget, value
Return value: void

f **textDisableRedisplay** . . . . *XawTextDisableRedisplay*

Arguments: ascii text wiget
Return value: void

f **textDisplay** . . . . *XawTextDisplay*

Arguments: ascii text wiget

Return value: void

f **textDisplayCaret** .... *XawTextDisplayCaret*

Arguments: ascii text wiget, boolean
Return value: void

f **textEnableRedisplay** .... *XawTextEnableRedisplay*

Arguments: ascii text wiget
Return value: void

f **textGetInsertionPoint** .... *XawTextGetInsertionPoint*

Arguments: ascii text wiget
Return value: XawTextPosition

f **textGetSelectionPos** .... *XawTextGetSelectionPos*

Arguments: ascii text wiget, varname for from position, varname for to position
Return value: void

f **textInvalidate** .... *XawTextInvalidate*

Arguments: ascii text wiget, from postion, to postion
Return value: void

f **textSearch** .... *XawTextSearch*

Arguments: ascii text wiget, scan direction, text block
Return value: XawTextPosition

f **textSetInsertionPoint** .... *XawTextSetInsertionPoint*

Arguments: ascii text wiget, position
Return value: void

f **textSetSelection** .... *XawTextSetSelection*

Arguments: ascii text wiget, from position, to position
Return value: void

f **textSetSelectionArray** .... *XawTextSetSelectionArray*

Arguments: ascii text wiget, selction array
Return value: void

f **textSinkMaxHeight** .... *XawTextSinkMaxHeight*

Arguments: ascii text wiget, height
Return value: int

f **textSinkMaxLines** . . . . *XawTextSinkMaxLines*

Arguments: ascii text wiget, height
Return value: int

f **textSinkSetTabs** . . . . *XawTextSinkSetTabs*

Arguments: ascii text wiget, tabs
Return value: void

f **textSourceRead** . . . . *XawTextSourceRead*

Arguments: ascii text wiget, position, varname for text block, length
Return value: XawTextPosition

f **textSourceScan** . . . . *XawTextSourceScan*

Arguments: ascii text wiget, position, scan type, scan direction, count, include (boolean)
Return value: XawTextPosition

f **textTopPosition** . . . . *XawTextTopPosition*

Arguments: ascii text wiget
Return value: XawTextPosition

f **textReplace** . . . . *XawTextReplace*

Arguments: ascii text wiget, from position, to position, text block
Return value: int

f **textUnsetSelection** . . . . *XawTextUnsetSelection*

Arguments: ascii text wiget
Return value: void

w **toggle** . . . . *toggle*

f **toggleChangeRadioGroup** . . . . *XawToggleChangeRadioGroup*

Arguments: toggle widget, NULL | toggle widget
Return value: void

f **toggleGetCurrent** . . . . *XawToggleGetCurrent*

Arguments: toggle widget
Return value: String

f **toggleSetCurrent** .... *XawToggleSetCurrent*

Arguments: toggle widget, radio data
Return value: void

f **toggleSetRadioData** .... *toggleSetRadioData*

Arguments: toggle widget, radio data
Return value: void

f **toggleUnsetCurrent** .... *XawToggleUnsetCurrent*

Arguments: toggle widget
Return value: void

w **viewport** .... *viewport*

f **viewportSetCoordinates** .... *XawViewportSetCoordinates*

Arguments: viewport Widget, x position, y position
Return value: void

f **viewportSetLocation** .... *XawViewportSetLocation*

Arguments: viewport Widget, xoffset, yoffset
Return value: void

A.4 Athena R5 Widgets and Commands

w **panner** .... *panner*

w **porthole** .... *porthole*

w **repeater** .... *repeater*

f **talk** .... *talk*

Arguments: panner, porthole, managed widget
Return value: void

w **tree** .... *tree*

f **treeForceLayout** .... *XawTreeForceLayout*

Arguments: tree Widget
Return value: void

A.5 Plotter Widget and Commands

w **axis** . . . . *atAxis*

w **barPlot** . . . . *atBarPlot*

f **barPlotAttachData** . . . . *AtBarPlotAttachData*

Arguments:	bar plot widget, floats
Return value:	void

w **labelAxis** . . . . *atLabelAxis*

f **labelAxisAttachData** . . . . *AtLabelAxisAttachData*

Arguments:	label axis wiget, strings
Return value:	void

w **linePlot** . . . . *atLinePlot*

f **linePlotAttachData** . . . . *AtLinePlotAttachData*

Arguments:	line plot widget, floats
Return value:	void

w **plotter** . . . . *atPlotter*

f **plotterGeneratePostscript** . . . . *AtPlotterGeneratePostscript*

Arguments:	file name, plotter widget, title, bounding box: x1, bounding box: y1, bounding box: x2, bounding box: y2, landscape
Return value:	void

w **textPlot** . . . . *atTextPlot*

w **xyAxis** . . . . *atXYAxis*

w **xyLinePlot** . . . . *atXYLinePlot*

f **xyLinePlotAttachData** . . . . *xyLinePlotAttachData*

Arguments:	xyLine plot widget, pair of floats (xfloat/yfloat)
Return value:	void

A.6 Tookit Widgets and Commands

f **addTimeOut** .... *XtAppAddTimeOut*

Arguments:	time interval in ms, tcl command
Return value:	void

f **action** .... *action*

Arguments:	widget, override | augment | replace, translatontable
Return value:	void

w **applicationShell** .... *applicationShell*

f **bell** .... *XBell*

Arguments:	widget to determine display, percent (-100 = off, 0 .. 100)
Return value:	void

f **callActionProc** .... *callActionProc*

Arguments:	widget, event type, action, args
Return value:	void

f **callCallbacks** .... *callCallbacks*

Arguments:	widget, callback, optional float and value
Return value:	void

f **callback** .... *callback*

Arguments:	widget, callback resource, none | exclusive | nonexclusive | popdown | position | positionCursor, string
Return value:	void

f **destroyWidget** .... *XtDestroyWidget*

Arguments:	widgets
Return value:	void

f **getClass** .... *getClass*

Arguments:	widget | None
Return value:	String

f **getResourceList** .... *getResourceList*

Arguments: widget, varname for output of reslist
Return value: int

f **getTypeOfAttribute** .... *getQTypeOfAttribute*

Arguments: widget, attribute
Return value: String

f **getValue** .... *getValue*

Arguments: widget, attribute
Return value: String

f **grabPointer** .... *XtGrabPointer*

Arguments: widget, true | false, pointer mode (sync | async), keyboard mode (sync | async), None | confine widget
Return value: int

f **grabKeyboard** .... *XtGrabKeyboard*

Arguments: widget, true | false, pointer mode (sync | async), keyboard mode (sync | async)
Return value: int

f **installAccelerators** .... *installAccelerators*

Arguments: destination, source, table
Return value: void

f **manageChild** .... *XtManageChild*

Arguments: widgets
Return value: void

f **mapWidget** .... *XtMapWidget*

Arguments: widgets
Return value: void

f **mergeResources** .... *mergeResources*

Arguments: widget to determine display, resource value pairs
Return value: void

w **overrideShell** .... *overrideShell*

f **popdown** .... *XtPopdown*

Arguments: shell widget
Return value: void

f **popup** . . . . . . *XtPopup*

Arguments: widget, none | nonexclusive | exclusive
Return value: void

f **popupSpringLoaded** . . . . . . *XtPopupSpringLoaded*

Arguments: shell widget
Return value: void

f **realize** . . . . . . *realizeWidgets*

Arguments:
Return value: void

f **removeAllCallbacks** . . . . . . *XtRemoveAllCallbacks*

Arguments: widget, callback name
Return value: void

f **setKeyboardFocus** . . . . . . *XtSetKeyboardFocus*

Arguments: from widget tree, to widget | None
Return value: void

f **setValues** . . . . . . *setValues*

Arguments: widget, resource value pairs
Return value: void

f **setWMProtocols** . . . . . . *XSetWMProtocols*

Arguments: Shell Widget, WM_TAKE_FOCUS | WM_SAVE_YOURSELF | WM_DELETE_WINDOW
Return value: Boolean

w **topLevelShell** . . . . . . *topLevelShell*

w **transientShell** . . . . . . *transientShell*

f **translateCoords** . . . . . . *XtTranslateCoords*

Arguments: widget, widget x, widget y, variable name for root x, variable name for root y
Return value: void

f **ungrabPointer** .... *XtUngrabPointer*

Arguments: widget with active pointer grab
Return value: void

f **ungrabKeyboard** .... *XtUngrabKeyboard*

Arguments: widget with active keyboard grab
Return value: void

f **unmanageChild** .... *XtUnmanageChild*

Arguments: widgets
Return value: void

f **unmapWidget** .... *XtUnmapWidget*

Arguments: widgets
Return value: void

f **unrealizeWidget** .... *XtUnrealizeWidget*

Arguments: widgets
Return value: void

f **disownSelection** .... *XtDisownSelection*

Arguments:
Return value: void

f **getSelectionValue** .... *XtGetSelectionValue*

Arguments: Tcl command
Return value: void

f **ownSelection** .... *ownSelection*

Arguments: string to be put into selection, NULL | loseSelection TCL-command, NULL | doneSelection TCL-command
Return value: Boolean

f **fetchBytes** .... *XFetchBytes*

Arguments: widget to determine display
Return value: String

f **convertSelection** .... *XConvertSelection*

Arguments: widget to determine display, target buffer number

Return value: void

f **storeBuffer** . . . . *XStoreBuffer*

Arguments: widget to determine display, String, buffer number (1-7)
Return value: void

f **fetchBuffer** . . . . *XFetchBuffer*

Arguments: widget to determine display, buffer number (1-7)
Return value: String

f **sV**
alias; for arguments see: *setValues*

f **gV**
alias; for arguments see: *getValue*

f **setIconPixmap** . . . . *setIconPixmap*

Arguments: Shell Widget (eg topLevel), pixmap file
Return value: void

f **changePixmap** . . . . *changePixmap*

Arguments: widget, resource, pixmap file
Return value: void

f **isWidget** . . . . *_name2Widget*

Arguments: widget name
Return value: int

w **layout** . . . . *layout*

A.7 Miscelaneous Additional Commands

f **ctime** . . . . *ctime*

Arguments: clock value
Return value: String

f **echo** . . . . *echo*

Arguments: strings

Return value: void

f **getChannel** . . . . *getChannel*

Arguments: None
Return value: int

f **quit** . . . . *kill*

Arguments: None
Return value: void

f **register** . . . . *doregister*

Arguments: stdin | stdout | stderr | xioerr, tcl-command
Return value: void

f **setCommunicationVariable** . . . . *setCommunicationVariable*

Arguments: variable name, length, tcl command to be executed
Return value: void

f **unregister** . . . . *unregister*

Arguments: stdin | stdout | stderr | xioerr
Return value: void

Index

References

[1] David Flanagan, *X Toolkit Intrinsics Reference Manual*, Third Edition, O'Reilly and Associates Inc., 1990.

[2] Arnaud Le Hors, *The X PixMap Format*, Part of the xpm distribution, 1991.

[3] Joel McCormack, Paul Asente and Ralph Swick, *X Toolkit Intrinsics – C Language Interface*, Massachusetts Institute of Technology, 1990.

[4] Gustaf Neumann, Stefan Nusser, *Wafe - An X Toolkit Based Frontend for Application Programs in Various Programming Languages* , Proc. USENIX Winter Conference, January 1993.

[5] Adrian Nye – Tim O'Reilly, *X Toolkit Intrinsics Programming Manual*, Second Edition, O'Reilly and Associates Inc., 1990.

[6] John K. Ousterhout, *Tcl: An Embeddable Command Language*, Proc. USENIX Winter Conference, January 1990.

[7] Ralph Swick – Terry Weissman, *X Toolkit Athena Widgets – C Language Interface*, Massachusetts Institute of Technology, 1990.