MIME Extensions for Mail-Enabled Applications: application/Safe-Tcl and multipart/enabled-mail

Nathaniel Borenstein, *First Virtual Holdings Inc.*
Marshall T. Rose, *Dover Beach Consulting, Inc.*

December, 1995

Status of this Memo

This document is a working draft.

Abstract

MIME [RFC-MIME] defines a format and general framework for the representation of a wide variety of data types in Internet mail. This document defines two new subtypes of MIME data, the application/Safe-Tcl and multipart/enabled-mail subtypes, for providing Enabled Mail [EM-MODEL] in the Internet community.

A table of contents appears at the end of this document.

1. Overview

Most electronic mail, even multimedia mail as standardized in MIME [RFC-MIME], is "passive" in the sense that the data are unidirectional. Textual, image, audio, or video data are simply displayed to the user, who reads, views, listens, or watches it and then must take specific action to initiate any response to the data, such as to reply to the originator, to replay the data, or to redistribute it to other recipients.

Less commonly used, but the subject of considerable research attention, has been "active messaging", in which the data delivered through the mail constitute a program in a well-specified language, allowing the program to be automatically evaluated on behalf of the recipient when the mail is "read." Researchers have demonstrated fascinating applications of this concept, and in recent years have shown that the critical problems of safety and portability can be solved in a straightforward manner [ATOMICMAIL].

This memo defines a standardized format for the interoperation of active messaging in the context of MIME. It defines a new language, "Safe-Tcl", based on the "Tcl" language [TCL]. It also defines a new application subtype, "application/Safe-Tcl", which may be used to tag a MIME entity (a mail body or body part) as being a program in the Safe-Tcl language. Additionally, this memo defines a new multipart subtype, "multipart/enabled-mail", for grouping together an interactive mail program and an arbitrary MIME entity to which it is related.

The reader should consult [EM-MODEL] for a description of the Enabled Mail model, which is not presented here. This memo also does not provide a tutorial in either the fundamental problems of safety and portability in active messaging, which the interested reader may find in [ATOMICMAIL]. Nor does this memo provide a tutorial in the Tcl language itself, for which the interested reader is referred to [TCL].

This memo assumes a basic familiarity with the syntax of Tcl, and defines Safe-Tcl in terms of its differences from standard Tcl. The resulting language is believed to be safe for use in Enabled Mail, according to the reasoning outlined in [ATOMICMAIL]. In particular, it is the intent of the Safe-Tcl language design that it should be essentially harmless to evaluate a Safe-Tcl program that comes from an unknown or hostile sender.

2. The application/Safe-Tcl content-type

The language defined in this memo is labelled as a MIME body or body part by the use of the "application/Safe-Tcl" content-type. Two mandatory parameters are defined for this content-type. The first parameter, "version", is a version number for the Safe-Tcl language itself. For the version of Safe-Tcl defined in this memo, the version value should be "7.3". The second parameter, "evaluation-time", is a string describing the intended time of evaluation for the Safe-Tcl program. Thus the Content-Type field might look something like this:

```
     Content-type: application/Safe-Tcl; version="7.3";
             evaluation-time=activation
```

The choice of "7.3" is indicative of the fact that the Safe-Tcl language, as described here, is derived from Tcl version 7.3. However, this should NOT be taken to indicate that arbitrary other versions of Tcl may be used with a corresponding change to the version parameter. If a future version of Safe-Tcl is ever defined, it will be formally specified and published as part of the MIME process. It is explicitly NOT the case that arbitrary versions of Tcl may be used with a suitably modified version parameter.

The evaluation-time parameter may have one of two values, "delivery" or "activation", which corresponds to the delivery-time and activation-time phases defined in [EM-MODEL]. A value of "activation" means that the program is intended to be evaluated whenever the user views the message, and may need to interact with the user. A value of "delivery" means that the program is intended to be evaluated upon final delivery to the user’s mailbox, and cannot interact directly with the user, though it may interact with user-supplied Safe-Tcl extensions.

Note that a MIME message that contains an application/safe-tcl entity with an evaluation-time of "delivery" is intended to be evaluated at delivery time. Such an entity will ONLY be evaluated, however, if it appears as either the top-level MIME content-type or directly inside a top-level multipart/enabled-mail entity. Otherwise, any nested MIME application/safe-tcl entity with an evaluation-time of "delivery" is ignored.

It is important for the Safe-Tcl programmer to note that Safe-Tcl programs do not automatically terminate evaluation at the end of the program evaluation. This is because the program may be event-driven, and may have created objects on the screen (e.g., command buttons) that are awaiting user input. If a Safe-Tcl program is intended to cause the Safe-Tcl process to terminate evaluation at the end of the program, it should end with the "exit" command.

3. The multipart/enabled-mail content-type

This memo defines a new subtype of the MIME multipart content-type, "multipart/enabled-mail". A multipart/enabled-mail entity has exactly two subparts, the first of which will be of arbitrary type, and the second of which will be of type application/Safe-Tcl (or some future language for Enabled Mail).

The intended semantics of multipart/enabled-mail, when viewed by a human reader, are as follows: If there is no application/Safe-Tcl interpreter available, or if the application/Safe-Tcl part has an evaluation-time of anything other than "activation", then the Safe-Tcl program should be skipped and the first part, an arbitrary MIME entity, should be displayed normally. If, however, the Safe-Tcl program has an evaluation-time of activation, and a Safe-Tcl interpreter is available, then the Safe-Tcl program should be evaluated with the first part of the multipart/enabled-mail object available to the Safe-Tcl program using the primitives defined in Section 4.3. Neither subpart is automatically displayed to the human reader, although the Safe-Tcl program may choose to display any or all of this information using the primitives defined in Section 4.5.

4. The Safe-Tcl Language

The syntax of Safe-Tcl is identical to the syntax of Tcl [TCL]. No syntactic constructs are changed. The only differences, therefore, between Tcl and Safe-Tcl is the set of available primitive functions and procedures. Safe-Tcl may be described as an "extended subset" of Tcl, in that the "dangerous" primitives in Tcl have been removed, while certain new primitives have been added.

It is assumed that the process evaluating a Safe-Tcl program will always have within it TWO interpreters, one for full Tcl (the trusted, or unrestricted, interpreter) and one for Safe-Tcl (the untrusted, or restricted, interpreter). A correct implementation will not allow a program being evaluated by the Safe-Tcl interpreter have access to the full Tcl interpreter except via the mechanisms defined as part of the Safe-Tcl language.

4.1. The Core Safe-Tcl Language

Because Tcl is an evolving language, it is not considered sufficient to describe Safe-Tcl completely in terms of differences from this base, as this may prove dangerously confusing if some future version of the language includes a potentially dangerous primitive not mentioned in the list of differences. Therefore, this memo provides a complete list of all the Safe-Tcl primitives "inherited" from standard Tcl. No other primitives should be provided by a Safe-Tcl interpreter. As a convenience to the reader, this memo will also list the standard Tcl primitives that were consciously omitted from

Safe-Tcl, but this list should not be considered exhaustive, in that any Tcl primitive that is not explicitly mentioned as being part of Safe-Tcl should be considered NOT to be part of Safe-Tcl.

In particular, the following standard Tcl commands are considered unsafe or inappropriate for use in Enabled Mail, and are NOT available within an untrusted interpreter:

> auto_execok, auto_load, auto_mkindex, auto_reset, cd, close, eof, exec, file, flush, gets, glob, open, pid, puts, pwd, read, seek, source, tell, time, unknown

The core set of standard Tcl commands which ARE a part of the Safe-Tcl language are:

> append, array, break, case, catch, concat, continue, error, eval, exit, expr, for, foreach, format, global, history, if, incr, info, join, lappend, lindex, linsert, list, llength, lrange, lreplace, lsearch, lsort, proc, regexp, regsub, rename, return, scan, set, split, string, switch, trace, unset, uplevel, upvar, while

It should be noted that the "exit" command in a Safe-Tcl interpreter need not have the same effect as the "exit" command in standard Tcl. In particular, if the Safe-Tcl interpreter is embedded in a larger process (such as a mail reader), the "exit" command should not terminate that process. Rather, its semantics should be those of terminating the execution of the current Safe-Tcl program, which may or may not entail terminating the process that is running it.

It is also probably a good idea to restrict the Tcl "rename" and "proc" commands to prevent Safe-Tcl programs from re-defining certain key Tcl primitives, notably "rename", "proc", and "exit". Although a Safe-Tcl program can’t do serious harm by renaming these primitives, it could work some very annoying and confusing mischief.

The core Safe-Tcl language also includes the following global variables from standard Tcl:

> errorCode, errorInfo

Other global variables might be defined as needed by a Safe-Tcl interpreter, but their presence should not be relied on.

In addition, Safe-Tcl includes additional built-in procedures and variables that are NOT part of standard Tcl, defined in the sections that follow. Some of these are available to all Safe-Tcl programs, while others are available only with certain values of the evaluation-time parameter or in certain user interface environments.

4.2. Universal Safe-Tcl Functionality

The following primitives are always part of the Safe-Tcl language.

SafeTcl_getconfdigdata -- 'ta 8n `"SafeTcl_getconfigdata key ?default? ?prompt?"`. To permit user customization of Safe-Tcl applications in the absence of any generalized file system access, Safe-Tcl includes a mechanism for associating a customization string with a key string. SafeTcl_getconfigdata returns the string value associated with the key, or generates an error otherwise. If the user has not previously specified the customization value (which might be performed in an implementation-specific manner), the Safe-Tcl interpreter engages the user in a dialog to obtain the value, which is also saved for future use. In this case, the optional second and third parameters provide a default value and a prompt to explain the nature of the data needed to the user. If the user HAS previously supplied a customization value to the user, then whether or not any user action is invoked by the SafeTcl_getconfigdata primitive is implementation-specific, but a suggested action is to use the previously-supplied value as a default and to ask the user to confirm that this is still the correct value. Note that if the evaluation-time is "delivery", then no user interaction is possible. In this case, if user-supplied data is available it is used; otherwise an error is generated. Each Safe-Tcl interpreter may implement particular special customization via SafeTcl_getconfigdata; the interpreter's documentation should be consulted for details.

SafeTcl_setconfigdata -- "'ta 8n `SafeTcl_setconfigdata key value`". This primitive may be used by a program to set a data for later retrieval with SafeTcl_getconfigdata, and returns the empty string.

SafeTcl_random -- 'ta 8n `"SafeTcl_random min max"`. This primitive returns a pseudo-randomly generated integer greater than or equal to the integer min and less than or equal to the integer max. If min is equal to max, that value is returned. If min is greater than max, an error is generated.

SafeTcl_genid -- 'ta 8n `"SafeTcl_genid"`. This primitive returns a string of less than 14 characters that is extremely likely to be unique for all time on the current machine. This makes it suitable for use as a temporary file name, a temporary variable name, or the portion of a Content-ID or Message-ID field to the left of the "@"-sign.

SafeTcl_loadlibrary -- "'ta 8n `SafeTcl_loadlibrary libname`". This primitive, when called from inside the untrusted interpreter, causes the trusted interpreter to look for a Safe-Tcl extension library. The manner by which this is found is implementation-specific. The library is evaluated in the trusted interpreter, and can use the "declareharmless" primitive to create extensions to the untrusted interpreter. (Note that Safe-Tcl libraries have no notion of version number; where this is needed, it can be incorporated into the library naming conventions.) The return value of this function is unspecified; an error is generated if the library cannot be loaded.

SafeTcl_runobj -- "'ta 8n `SafeTcl_runobj hash code`". This primitive, when called from the trusted interpreter, takes an MD5 and a base64 string. It decrypts the base64 string, checks that the MD5 hash is valid, and if so, evaluates it as Tcl code.

SafeTcl_loadobj -- "'ta 8n `SafeTcl_loadobj filename`". This primitive when called from the trusted interpreter, takes a file name as its only argument. It opens the file and reads the first two lines. If " -objfile " is in the first line of the file, it passes the second line as the MD5 hash and the rest of the file as the base64 string to SafeTcl_runobj. If " -objfile " isn't in the first line, it assumes it's normal source and sources the file.

Additionally, Safe-Tcl always defines a global variable that indicates the current evaluation-time context:

SafeTcl_evaluation_time -- A string that is set to either "delivery" or "activation" to indicate the current evaluation-time context.

4.3. Additional Messaging Functionality

The following primitives are always part of the Safe-Tcl language when used in the context of Enabled Mail, but may not be present if the language is adopted for use outside of this context.

Note that four of these primitives (SafeTcl_getheader, SafeTcl_getheaders, SafeTcl_getbodyprop, and SafeTcl_getparts) may make implicit reference to a MIME message. Each of these primitives has an optional parameter, "?body?", which contains a MIME entity. If this optional parameter is not supplied (or is supplied but is empty), then, in the case of evaluation-time "delivery", the "?body?" parameter defaults to the entire MIME message; otherwise, in the case of evaluation-time "activation", if the Safe-Tcl program is part of a multipart/enabled-mail MIME entity, then the "?body?" parameter defaults to the OTHER part of that multipart/enabled-mail entity. Otherwise, if an explicit "?body?" parameter is not present, an error is generated.

SafeTcl_getaddrs --'ta 8n `"SafeTcl_getaddrs string"`. This primitive returns a list, each element of which is a string containing an electronic mail address found in the parameter.

SafeTcl_getaddrprop -- 'ta 8n `"SafeTcl_getaddrprop address property"`. This primitive returns the specified property from the address string, which is an address specification in RFC 822/1123 format.

Properties are:

Property	*Returns*	*Description*
proper	string	official 822 rendering, e.g. "phrase <local@domain>"
friendly	string	user-friendly rendering (see Appendix C)
address	string	local@domain rendering
phrase	string	the phrase part
local	string	the local part
mymbox	integer	"1" if this is the recipient’s address, as determined by local configuration, "0" otherwise.
domain	string	the domain part
verify	string	"" if the address/domain appear valid ERROR -- if address is invalid non-empty string -- no definite answer, explanation given in the string

If the string can not be parsed as an electronic mail address, an error is generated . If the address parameter is the empty string, the current user’s address is used.

SafeTcl_getdateprop -- ’ta 8n `"SafeTcl_getdateprop date property"`. This primitive returns the specified property from the date/time string, which is a date specification in RFC 822/1123 format. Properties are:

Property	*Returns*	*Description*
sec	integer	seconds of the minute
min	integer	minutes of the hour
hour	integer	hours of the day (0-23)
wday	integer	day of the week (Sun=0)
day	string	day of the week (3 char abbreviation)
weekday	string	day of the week
sday	integer	day of the week known? (1=explicit 0=implicit, -1=unknown)
mday	integer	day of the month
yday	integer	day of the year
mon	integer	month of the year
month	string	month of the year (3 char abbreviation)
lmonth	string	month of the year
year	integer	year (all digits, e.g. 1993)
zone	integer	timezone in minutes
tzone	string	timezone string
szone	integer	timezone known? (1=explicit, 0=implicit, -1=unknown)

```
date2local string  coerce date to local timezone
date2gmt   string  coerce date to GMT
dst        integer daylight savings in effect?
rclock     integer seconds prior to current time
proper     string  official 822 rendering
```

If the string can not be parsed as a date/time, an error is generated. If the date parameter is the empty string, the current date and time are used.

SafeTcl_getheader -- 'ta 8n `"SafeTcl_getheader field ?body?"`. This primitive returns the value of a field in the message's (or MIME entity's) headers. If the field is not present in the headers, the empty string is returned. If there are multiple headers with the same field name, the values will be concatenated for the fields "To", "cc", "bcc", "Reply-To", Resent-To", "Resent-cc", "Resent-bcc", and "Resent-Reply-To". Otherwise only the first occurrence, if any, will be returned.

SafeTcl_getheaders -- 'ta 8n `"SafeTcl_getheaders ?body?"`. This primitive returns a list, each element of which identifies a header field contained in the message (or MIME entity). Each element is a list containing two string elements, the first of which is a header field name, and the second of which is a header field body. If the MIME entity contains multiple headers with the same field name, each occurrence will appear as a different element.

SafeTcl_makebody -- This primitive is invoked as either 'ta 8n `"SafeTcl_makebody multipart-content-type ?-id string? ?-parameter string? ?-description string? body ..."` or 'ta 8n `"SafeTcl_makebody content-type ?-id string? ?-parameter string? ?-description string? value ?encoding?"`, and returns a MIME entity of the specified content-type. The first invocation syntax is used for multipart entities, the content-type begins with "multipart/", and each "body" parameters (of which any number may be present) must each be a complete MIME entity, as returned, for example, by previous calls to SafeTcl_makebody). The second invocation syntax is used for other content-types, and one or two parameters are present, the (encoded) value of the MIME body being constructed, and the transfer encoding (e.g., "base64"), respectively. If the content-type parameter is the empty string, "text/plain" is assumed. There may be zero or more occurrences of "-parameter string" to indicate whatever parameters are associated with the content- type. Note that for multipart bodies, a boundary parameter should NOT be supplied; SafeTcl_makebody will derive one as needed. There may be at most one occurrence of "-description string" to specify the Content-Description field. There may be at most one occurrence of "-id string" to specify the Content-ID field. (If this sequence is not present, a Content-ID will be generated automatically.)

SafeTcl_getparts -- ’ta 8n "`SafeTcl_getparts ?body?`". This primitive returns a list, each element being a list that identifies a MIME entity contained within the body parameter. Each of these lists consists of a numeric identifier, a content-type, a content-description string (empty if there is none), and an ESTIMATE of the size of the message, in kilobytes. The return value is constructed by a pre-order traversal of the body, with the numeric identifier of a parent being used as the prefix for its subordinates. If a MIME entity is a message/external-body content, then its subordinate, the external content is also present in the list returned. For example, if the structure of a message were:

```
multipart/mixed
  text/plain
  multipart/digest
    message/rfc822
  message/external-body
    audio/basic
```

then the list returned would have this structure:

```
{{"1" "multipart/mixed" "A bunch of stuff" 242}
   {"1.1" "text/plain" "Introduction" 1}
   {"1.2" "multipart/digest" "Today’s news" 4}
     {"1.2.1" "message/rfc822" "A word from Bill" 3}
   {"1.3" "message/external-body" 236}
     {"1.3.1" "audio/basic" "Many words from Bill" 235}}
```

SafeTcl_getbodyprop -- "’ta 8n `SafeTcl_getbodyprop part property ?body?`". This primitive returns a string containing the value of the specified property for the body part specified by the first and third parameters. The part specification may either be an index as returned by SafeTcl_getparts, or a content-id value (surrounded in angle brackets). Properties are:

```
Property   Returns Description
```

all	string	complete MIME entity, with headers and body, suitable, e.g., for SafeTcl_displaybody or SafeTcl_getparts.
descr	string	value of Content-Description field
encoding	string	Content-Transfer-Encoding value (or "7bit" if none specified)
headers	string	all header lines
id	string	value of Content-ID field
parms	list	each element a parameter from the Content-Type field, given as a list of two items {paramname paramvalue}
size	integer	length of (encoded) value
type	string	value of Content-Type field (without parameters)
value	string	body, possibly encoded

If the body part identified is a subordinate to a message/external-body content, and the property specified is either "all", "size", or "value", then the appropriate access-method may be invoked. For the untrusted interpreter, only the "anon-ftp" access-method is supported. In addition, the "local-file" access-method is supported for the trusted interpreter. If the access-method is missing or unsupported, then an error (either "missing access-type" or "unsupported access-type") is generated. Note that if the access-method is mail-server, then a "mail-server access-type" error is generated, and the caller may invoke SafeTcl_sendmessage (or MIME_sendmessage) accordingly. In this case, the message/external-body should be examined for these parameters: server, subject, and body. Finally, note that an implementation may support content caching in order to avoid unnecessarily retrieving an external content. In this case, the content cache is consulted to see if the content identified by the Content-ID field is available locally, and if so, the access-method is ignored -- the content stored in the cache will be used instead. If an implementation chooses to implement content caching, it must do so in a manner transparent to SafeTcl_getbodyprop; further, it should provide the trusted interpreter with a mechanism for disabling use of the content cache.

SafeTcl_encode -- "’ta 8n `SafeTcl_encode encoding data`". This primitive takes the specified data and encodes it according to the MIME encoding specified by the encoding parameter, (e.g., "base64") and returns a string that is the encoded data.

SafeTcl_decode -- "’ta 8n `SafeTcl_decode encoding data`". This primitive takes the specified encoded data and decodes it according to the MIME encoding specified by the encoding parameter, (e.g., "base64") and returns a string that is the decoded data.

SafeTcl_sendmessage -- ’ta 8n `"SafeTcl_sendmessage -to <addrlist> -subject <string> -body <body> ?-cc <addrlist>? ?-auxheader <name> <value>? ?-queue? ?-resent? ?-atleastone?"`. This primitive sends a message and returns the empty string on success. It takes a variable number of key/value parameters, three of which are required. The required "-to <addrlist>" parameter specifies the primary recipients, as a string containing one or more electronic mail addresses RFC 822/1123 format (e.g., with commas separating multiple addresses). The required "-subject <string>" parameter specifies the subject of the mail being sent. The required "-body <body>" parameter specifies the mail body, which is the value returned by a SafeTcl_makebody call. The optional "-cc <addrlist>" parameter specifies the secondary recipients, again as a single string of RFC 822/1123 electronic mail addresses. The optional "-auxheader <name> <value>" parameter (of which any number may be present) specifies additional headers which may be added to the mail message, e.g., "-auxheader Reply-to: nsb@nsb.fv.com". The optional "-queue" switch indicates that it is preferrable that the message be queued for later delivery (rather than immediate delivery). The optional "-resent" switch indicates that the message is actually being resent. By default, in the "activation" evaluation-time, the Safe-Tcl interpreter will offer the user the opportunity to edit the message before it is delivered, and ask for confirmation from the user before the message is sent. It is acceptable for an interpreter to provide the user with a mechanism for selectively disabling this confirmation process. The optional "-atleastone" switch indicates that errors with some of the destination addresses should be ignored if at least one of them works correctly.

Two additional optional switches may be available on some platforms: "?-originator address?" sets the envelope originator address (if different from -auxheader From address), and "-auxheader Dcc addresses" may be used to add addresses to the envelope, but not the headers.

SafeTcl_printtext -- ’ta 8n `"SafeTcl_printtext ?txt?"`. This primitive sends plain textual data to a locally-available printer, returning the empty string on success. The "?text? paremeter defaults to the body of the current message. By default , in the "activation" evaluation-time , the Safe-Tcl interpreter will ask for confirmation from the user for the text to be printed. It is acceptable for an interpreter to provide the user with a mechanism for selectively disabling this confirmation process.

SafeTcl_savemessage -- "’ta 8n `savemessage type ?destination?`". This primitive appends the message to either a mailbox or a folder, as specified by the first parameter, which should be either "mailbox" or "folder", returning the empty string on success. If the second parameter is the empty string, the recipient’s default mailbox or folder is used. Note that the concepts of mailbox and folder are highly implementation-specific, and, as such, the behavior of this primitive may also be user-customizable to deal with different mailbox and folder formats. By default, in the "activation" evaluation-time, the Safe-Tcl interpreter will ask for confirmation from the user before the message is saved. It is acceptable for an interpreter to provide the user with a mechanism for selectively disabling this confirmation process.

4.4. Additional Delivery-Time Functionality

When a Safe-Tcl program is delivered in a mail message with the evaluation-time parameter given as "delivery", then no interaction with a user is possible. In this context, two additional global variables are available:

SafeTcl_Originator -- A string containing the originator of the message, as indicated by the envelope.

SafeTcl_Recipient -- A string containing the recipient of the message, as indicated by the envelope.

4.5. Additional Activation-Time Functionality

When a Safe-Tcl program is received in a mail message with the evaluation-time parameter given as "activation", the mail message is intended to be run in an interactive setting. Several additional Safe-Tcl procedures may become available in this context, some of which are only available in certain user interface contexts.

In a Safe-Tcl application outside of Enabled Mail, these primitives may also be present if and only if there is a user with whom the program can interact.

4.5.1. User Interaction Models: SafeTcl_InterfaceStyle

Second only to safety as a critical issue for an active messaging language is the question of user interface capabilities. Since the language needs to be able to work in a wide variety of hardware and software environments, it is difficult to avoid a "lowest common denominator" user interface. Safe-Tcl addresses this problem by providing a few lowest common denominator primitives, and then by providing a mechanism by which the availability of well-defined packages of more advanced user interface mechanisms can be made known to a Safe-Tcl program at runtime.

Each Safe-Tcl interpreter must provide a global variable, SafeTcl_InterfaceStyle, a list, each item of which indicates a user interface extension set that is available. At a minimum, the "generic" interface is always available.

Thus if a Safe-Tcl interpreter supported both the "foo" and "bar" user interface extensions, it would set SafeTcl_InterfaceStyle to some permutation of {generic foo bar}.

It is expected that a common approach to writing Safe-Tcl programs will be to include multiple versions of user interface functions, e.g.:

```
    if {[lsearch $SafeTcl_InterfaceStyle "Tk3.*"]} {
        do_Tk_style_interaction
    } else {
        do_generic_style_interaction
    }
```

The items in the SafeTcl_InterfaceStyle should all be interpreted in a case-insensitive manner.

4.5.2. Generic User Interaction

The following user interaction primitives are available in the "generic" interface style, and hence are available to all interactive Safe-Tcl programs.

SafeTcl_displaytext --'ta 8n `"SafeTcl_displaytext text"`. This primitive shows the given text to the user, returning zero on success. The string specified may be of arbitrary length, so consideration must be given to scrolling or pagination as necessary.

SafeTcl_displayline -- 'ta 8n `"SafeTcl_displayline text"`. This primitive shows the given text to the user, returning zero on success. The string specified is a single line of text.

SafeTcl_gettext -- 'ta 8n `"SafeTcl_gettext prompt ?default?"`. This primitive obtains an arbitrary body of text from the user, which is returned as a Tcl string. The first parameter is used as a prompting string to solicit the text from the user, while the optional second parameter is the default to be offered.

SafeTcl_getline -- 'ta 8n `"SafeTcl_getline prompt ?default?"`. This primitive obtains a single line of text from the user, which is returned as a Tcl string. The first parameter is used as a prompting string to solicit the text from the user, while the optional second parameter is the default to be offered.

SafeTcl_displaybody -- 'ta 8n `"SafeTcl_displaybody ?-background? ?body?"`. This primitive causes a MIME entity to be displayed to the user, returning the empty string on success. The "?body?" parameter (which defaults to the current message) should be a string containing a complete MIME entity (e.g., returned by SafeTcl_makebody, or by SafeTcl_getbodyprop when asked for the "all" property). The manner in which the entity is displayed, and the set of MIME types that are supported, is implementation-specific. If the "-background" option is specified, the MIME entity will be displayed in the background, that is, in parallel with the ongoing Safe-Tcl program which will not wait for the completion of the display. (This is particularly useful for audio and other temporal media.) If a Safe-Tcl implementation does not support such parallel execution, then the use of -background generates a "No Background Display" error.

Note that the five primitives SafeTcl_displaytext, SafeTcl_displayline, SafeTcl_gettext, SafeTcl_getline, and SafeTcl_displaybody, constitute the entire "generic" user interface of the core Safe-Tcl language. Additional user interface capabilities may be indicated using the always-present global variable SafeTcl_InterfaceStyle, as described above. However, these five primitives are guaranteed to be available for every Safe-Tcl implementation, and can be used either to write "lowest common denominator" user interfaces or to provide a backup user interface when the SafeTcl_InterfaceStyle variable indicates that no recognized user interface extensions are available.

4.5.3. X11 Interaction: Interface Style Tk3.6

This document defines the use of a single user interface extension set, corresponding to a large subset of Tk, the X Window System extensions for Tcl. The availability of this user interface capability is declared by the inclusion of the string "Tk3.6" in the SafeTcl_InterfaceStyle variable. The choice of "3.6" is indicative of the fact that the Tk primitives described here are derived from Tk version 3.6. However, this should NOT be taken to indicate that arbitrary other versions of Tk may be used with a corresponding change to the SafeTcl_InterfaceStyle string. If a future version of the Tk interface style for Safe-Tcl is ever defined, it will be formally specified and published as part of the MIME process. It is explicitly NOT the case that arbitrary versions of Tk may be used with a suitably modified InterfaceStyle value.

As with the core Safe-Tcl language, the Tk extensions will be described in terms of differences from the standard Tk3.6 language. Since Tk does not extend the basic syntax of the language, all that needs to be specified is the set of available primitives.

Only two Tk primitives are omitted from Safe-Tcl, namely the "send" and "toplevel" primitives. Toplevel is replaced by "mkwindow", defined below.

All of the other core Tk procedures and functions are retained. In particular, the COMPLETE set of functions to be define by the "Tk3.6" interface style for Safe-Tcl is as follows:

> after, bind, button, canvas, checkbutton, destroy, entry, focus, frame, grab, label, lineto, listbox, lower, menu, menubutton, message, moveto, option, pack, place, raise, radiobutton, scale, scrollbar, selection, text, tk, tk_bindForTraversal, tk_firstMenu, tk_getMenuButtons, tk_invokeMenu, tk_mbButtonDown, tk_mbPost, tk_mbUnpost, tk_menuBar, tk_menus, tk_nextMenu, tk_nextMenuEntry, tk_traverseToMenu, tk_traverseWithinMenu, tkwait, update, winfo, wm

The "mkwindow" primitive is added to replace the standard Tk "toplevel" command. It is defined as follows:

> **mkwindow** -- "'ta 8n `mkwindow`". Creates a new toplevel window, already "decorated" to make it obvious to the user that it is a window belonging to an untrusted program, and returns the window's name as a Tcl string.

The standard Tcl implementation uses the same code for "frame" and "toplevel"; care must be taken to ensure that "rename frame toplevel" does not restore the "toplevel" functionality.

Additionally, the Tk "grab" command must be modified to disable the -global switch, which could lock up a user's machine. And the Tk "wm" command must be modified to prohibit the "overrideredirect" option, and so that the "wm geometry" option can not be used to cause Safe-Tcl's window decorations to be placed off-screen. It has also proven desirable to place a limit on the number of times that window geometries can be changed, as annoying programs can otherwise cause windows to prance around uncatchably on the user's screen.

There are several ways that a Tk program can try to "freeze" the user's screen, possibly requiring a system reboot if the user can't get to the machine via the network and kill the Tk process. For this reason, it is strongly recommended that any implementation of Safe-Tcl that uses the Tk interface style should always provide a "master control window" that gives the user an easy way to kill the Safe-Tcl process. Such a master control window should endeavor to always keep itself visible on the user's screen.

Some other standard Tk commands, notably destroy, pack, and place, may require user-invisible modifications to ensure that they do not offer a mechanism to subvert the window-decoration scheme implemented by mkwindow.

An additional command is provided in this context, to give access to the list of X11 fonts available on the current display:

> **SafeTcl_ListFonts** -- 'ta 8n `"SafeTcl_ListFonts ?pattern?"`. This primitive returns a list of fonts available on the current X11 display. If a pattern is supplied, only fonts matching that pattern are returned. If no fonts match the pattern, the empty string is returned.

5. Extensions to the Safe-Tcl Environment

It is relatively easy to extend the Safe-Tcl environment, though this should be done with great caution, since the introduction of a new Safe-Tcl command always carries with it important security implications. In particular, whenever a new command is added, the author should consider whether this command could be used by malicious parties to cause any harm.

To extend Safe-Tcl, one writes a procedure in full Tcl, to be interpreted by the trusted Tcl interpreter. A particular command may then be made available in the SafeTcl environment using the "declareharmless" primitive, as described in the following section. Users may insert their own initialization code in an implementation-specific configuration file. This code will be evaluated by the trusted interpreter, but may define extensions to the untrusted interpreter using "declareharmless" or may evaluate expressions in the untrusted interpreter using "restrictedeval".

6. Extensions to the *Trusted* Tcl Interpeter

In order to permit the core Safe-Tcl language to have extensibility and minimal support to sending message and generating paper output, it is necessary that certain procedures be added to the trusted interpreter. Note that these commands should NOT be added to the Safe-Tcl interpreter, as this would not be considered safe. (One could imagine a mail message that automatically generates hate mail in the recipient's name, or a message that maliciously wastes printer resources or sabotages a programmable printer.)

The following procedures are to be included in the TRUSTED interpreter that is accessible to Safe-Tcl programs via the extension mechanisms previously described:

MIME_sendmessage -- 'ta 8n `"MIME_sendmessage ..."`. This primitive is identical to SafeTcl_sendmessage, except that user confirmation is not required.

MIME_printtext -- 'ta 8n `"MIME_printtext ..."`. This primitive is identical to SafeTcl_printtext, except that user confirmation is not required.

MIME_savemessage -- "MIME_'ta 8n `savemessage ...`". This primitive is identical to SafeTcl_savemessage, except that user confirmation is not required.

declareharmless -- "'ta 8n `declareharmless command`". This primitive makes the specified command available in the untrusted interpreter. All parameters will be evaluated in the untrusted interpreter.

CRITICAL WARNING: Safe-Tcl programmers should exercise extreme caution in the use of declareharmless. The effect of "declareharmeless your-command" is to make all the power of "your-command" available, on YOUR computer and with YOUR system privileges, to ANYONE who can send you email. Tcl programs that are exported to the untrusted interpreter in this manner must be carefully designed to avoid making any dangerous functionality available in the untrusted interpreter. In particular, they must NEVER evaluate their parameters as arbitrary Tcl expressions. More generally, such commands should be designed to be as specialized and restrictive as possible. The declareharmless primitive should NEVER be used without asking how a malicious or hostile third party might exploit the new functionality being provided.

restrictedeval -- "'ta 8n `restrictedeval program`". This command evaluates the given Tcl program in the untrusted environment in a global context, giving the trusted interpreter access to the current state of the untrusted interpreter.

MIME_ConfirmAction -- "'ta 8n `MIME_ConfirmAction prompt yesstr nostr inspectstr inspectdata`". This primitive causes the user to be prompted to confirm a potentially dangerous action, such as sending mail. The prompt parameter gives the question that will be asked, while yesstr and nostr are the strings that are offered to allow the user to permit or disallow the action in question. The inspectstr parameter is the string that the user may choose in order to inspect the data given in inspectdata. MIME_ConfirmAction is intended for use to implement, for example, SafeTcl_sendmessage given the MIME_sendmessage primitive in the trusted interpreter, but it is also useful for other Safe-Tcl extensions that may require user confirmation. It is defined in the trusted interpreter so that it may not be circumvented by any code running in the untrusted interpreter.

An additional variable may be provided in the trusted interpreter to indicate that additional services have been applied prior to the invocation of the Safe-Tcl interpreter.

Note that this variable exists within the trusted interpreter, and can not be set by untrusted programs.

SafeTcl_Services -- A Tcl array describing additional services which were applied to the message prior to the invocation of the interpreter. Each element in the array is a string. At present, two array elements are defined:

authentication - which indicates that the value of this element, if present and not the empty string, was authenticated as the originator of the message, and that the authentication process included a message integrity check. Extension code running in the trusted interpreter can use this value to permit, for example, a greater range of actions to programs from authenticated and trusted sources.

privacy - which, if present and not the empty string, indicates that the message transmitted in ciphertext, and was deciphered prior to being passed to the Safe-Tcl interpreter

7. Notes To Implementors

Implementation details are usually considered beyond the scope of a specification such as this one. However, the extremely sensitive nature of a Safe-Tcl interpreter, and the safety issues entailed in the implementation of such an interpreter, suggest that some advice to implementors might be useful here.

1. Be careful how you implement any user interface code that asks for confirmation of potentially dangerous actions, e.g. in MIME_sendmessage or MIME_printtext. In particular, such code should always be evaluated in the trusted interpreter, to prevent hostile programs from "reverse engineering" your implementation and short-circuiting the confirmation code. (A more radical alternative is to prohibit the use of the Tcl primitives proc or rename to redefine any built-in Safe-Tcl primitives or any Safe-Tcl procedures defined for the confirmation process itself.)

2. A Safe-Tcl interpreter should ensure that there is ALWAYS some indication on the screen that an untrusted program is being run. A suggested mechanism is to reserve, as a "status line" the top or bottom line of each terminal or window in which Safe-Tcl is running. That line should indicate that the program is not to be trusted with sensitive information. This will help to prevent a clever Safe-Tcl program from fooling the user into supplying a password, e.g. by spoofing a login program.

3. A Safe-Tcl interpreter that runs on a video display terminal or terminal emulator should beware of permitting a Safe-Tcl program to send escape codes to the terminal. Some terminals can be programmed, using binary escape codes, to send data to the terminal which is then sent back to the host computer, and might be used to "break out" of the Safe-Tcl environment. Safe-Tcl interpreters that run on such terminals might wish to render into printable characters all lines that are displayed with such primitives as SafeTcl_displaytext and SafeTcl_displayline, thus inhibiting the transmission of raw escape codes to the terminal.

4. Special care must be paid to all uses of declareharmless, as this is an obvious place in which security holes may be introduced.

Security Considerations

Active messaging, in which programs are sent through the mail to be evaluated automatically or semi-automatically on behalf of the recipient, is an area fraught with potential security problems. Accordingly, the most important aspect of the design of the application/Safe-Tcl MIME type was the care that was paid to defining an active messaging language that was capable of safe implementation.

Despite this care, there remain two potential pitfalls that could cause the application/Safe-Tcl type to become a vehicle for security problems, and all implementors and administrators should be aware of these problems:

1. Implementation bugs. No matter how much care is paid to the design of a language for active messaging, interpreters of such a language will remain as vulnerable to security-compromising bugs as any other network services. Just as the Internet worm was able to exploit bugs in the finger and sendmail programs, so too bugs in a Safe-Tcl interpreter might be exploited by future sociopaths. This does not argue against the concept of Safe-Tcl, but suggests that system administrators must be clear about the difference between a safe language and a correct implementation of a safe language, and should INSTALL ONLY THOSE SAFE-TCL INTERPRETERS THAT COME FROM EXTREMELY TRUSTED SOURCES.

2. Poorly-conceived extensions or supersets. Safe-Tcl is, by design, a highly restricted language. It is very easy to add extensions that will make it more powerful, but such extensions can easily have the effect of undoing all the security-consciousness that went into the original design of the language. (Such extensions won’t exactly promote interoperability, either, another good reason to avoid them.) Administrators should beware of installing software that claims to implement a superset of Safe-Tcl, as the basic Tcl language is not itself safe for sending through the mail. Users and administrators alike must exercise care in the use of the Safe-Tcl extension mechanisms. When in doubt, simply don’t install an extension to Safe-Tcl.

In addition, implementations should also consider limiting the ability of Safe-Tcl programs to consume system resources. For example, malicious or buggy programs might create an infinite loop or consume enough window system resources to prevent any other work from being done. Resources that might be limited by an interpreter include, but are not limited to, CPU time, number of windows, number of mail messages sent, function call stack space, heap space, and so on.

Authors’ Addresses

For more information, the author of this document may be contacted via Internet mail:

Nathaniel S. Borenstein
25 Washington Avenue
Morristown, NJ 07960
US

Phone: +1 201 540 8967
Fax: +1 201 993 3032
Email: nsb@nsb.fv.com

Marshall T. Rose
Dover Beach Consulting, Inc.
420 Whisman Court

Mountain View, CA 94043-2186
US

Phone: +1 415 968 1052
Fax: +1 415 968 2510
Email: mrose@dbc.mtview.ca.us

Acknowledgements

This document reflects the input and ideas of many researchers and developers who have worked in the field of active messaging over the last two decades. Particularly helpful in the drafting of this document have been Dave Crocker, Karl Lehenbauer, John Ousterhout, Rich Salz, and Allan Shepherd.

Special thanks are due to John Ousterhout, for the design and implementation of Tcl and Tk.

References

[RFC-MIME] Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1341, June, 1992.

[ATOMICMAIL] Borenstein, Nathaniel S., "Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work", in Proceedings of CSCW '92 Conference, Toronto, Ontario, November, 1992.

[TCL] Ousterhout, John, Tcl and the Tk Toolkit. Addison-Wesley, 1993 (to appear).

[EM-MODEL] Rose, M., and N. Borenstein, "A Model for Enabled Mail (EM)", draft in preparation, May, 1993.

Appendix A: Examples

A. 1. Usage Example: Delivery-Time Enabled Mail

Here is a brief example of a program that might be evaluated during the delivery evaluation-time.

```
Content-Type: application/safe-tcl; version="7.3";
      evaluation-time=delivery

SafeTcl_sendmessage \
        -to $SafeTcl_Originator \
        -subject "Delivery Notification for $SafeTcl_Recipient" \
        -body [SafeTcl_makebody "text/plain" \
        [SafeTcl_getheader "Message-ID"]]
```

This simply sends a message to the originator indicating that the incoming message crossed the delivery slot for the recipient.

A. 2. Usage Example: Activation-Time Enabled Mail

Here is a brief example of a program that might be evaluated during the activation evaluation-time.

```
Content-Type: application/safe-tcl; version="7.3";
      evaluation-time=activation

proc ordershirt {} {
    SafeTcl_sendmessage  -to tshirts@nowhere.really \
            -subject "Shirt request" \
            -body [SafeTcl_makebody "text/plain" \
                         [SafeTcl_getline \
                              "What size t-shirt do you wear?" \
                              "medium"] "" ]
    exit
}

if {[lsearch $SafeTcl_InterfaceStyle "Tk3.*"] >= 0} {
    set foo [mkwindow]
    message $foo.m -aspect 1000 \
        -text "Click below if you want a free Bill Clinton t-
shirt!"
    button $foo.b -text "Click here for free shirt!" \
        -command {ordershirt}
    button $foo.b2 -text "Click here to exit without ordering" \
         -command exit
    pack append $foo $foo.m {pady 20} $foo.b {pady 20} \
         $foo.b2 {pady 20}
} else {
    set ans [string index \
                     [SafeTcl_getline  \
                     "Do you want a free Clinton t-shirt? "  \
                     "No"] \
                   0]
    if {$ans == "y" || $ans == "Y"} {
        ordershirt
    }
    exit
}
```

The above program (which will use the Tk3.6 interface style if it is available, and will otherwise use the default style) will offer the user the opportunity to order a free t-shirt.

Appendix B: Summary of Safe-Tcl Primitives

This document defines several extensions to Tcl, some of which are available in different contexts. Some are extensions that are always present in a Safe-Tcl interpreter. Others are only present in a Safe-Tcl interpreter used in an Enabled Mail context, or only when the program is running at activation time, with a user present. Others are only present in the TRUSTED (unrestricted) interpreter that is the companion to the UNTRUSTED (restricted) Safe-Tcl interpreter. This appendix summarizes the extensions defined here and their domains of applicability.’ta 8n

Name	Type	Interpreter	Applicability	
SafeTcl_getconfigdata	command	Untrusted	Universal	
SafeTcl_setconfigdata	command	Untrusted	Universal	
SafeTcl_random	command	Untrusted	Universal	
SafeTcl_genid	command	Untrusted	Universal	
SafeTcl_loadlibrary	command	Untrusted	Universal	
SafeTcl_evaluation_time	variable	Untrusted	Universal	
SafeTcl_getaddrs	command	Untrusted	Messaging	
SafeTcl_getaddrprop	command	Untrusted	Messaging	
SafeTcl_getdateprop	command	Untrusted	Messaging	
SafeTcl_getheader	command	Untrusted	Messaging	
SafeTcl_getheaders	command	Untrusted	Messaging	
SafeTcl_makebody	command	Untrusted	Messaging	
SafeTcl_getparts	command	Untrusted	Messaging	
SafeTcl_getbodyprop	command	Untrusted	Messaging	
SafeTcl_encode	command	Untrusted	Messaging	
SafeTcl_decode	command	Untrusted	Messaging	
SafeTcl_sendmessage	command	Untrusted	Messaging	
SafeTcl_printtext	command	Untrusted	Messaging	
SafeTcl_savemessage	command	Untrusted	Messaging	
SafeTcl_Originator	variable	Untrusted	Messaging	
SafeTcl_Recipient	variable	Untrusted	Messaging	
SafeTcl_displaytext	command	Untrusted	Activation-time	
SafeTcl_displayline	command	Untrusted	Activation-time	
SafeTcl_gettext	command	Untrusted	Activation-time	
SafeTcl_getline	command	Untrusted	Activation-time	
SafeTcl_displaybody	command	Untrusted	Activation-time	
mkwindow	command	Untrusted	Activation w/Tk	
SafeTcl_listfonts	command	Untrusted	Activation w/Tk	
SafeTcl_loadobj	command	Trusted	Universal	
SafeTcl_runobj	command	Trusted	Universal	
MIME_ConfirmAction	command	Trusted	Universal	
MIME_sendmessage	command	Trusted	Messaging	
MIME_printtext	command	Trusted	Messaging	
MIME_savemessage	command	Trusted	Messaging	
declareharmless	command	Trusted	Universal	******
restrictedeval	command	Trusted	Universal	
SafeTcl_Services	variable	Trusted	Messaging	

******The "declareharmless" command should be used with extreme caution, and only by those who are confident that they fully understand the security implications of its use, as described earlier in this document.

Appendix C: User-Friendly Renderings

When displaying an RFC 822 address, a user-friendly rendering may be preferred. In practice, an RFC 822 address usually appears in one of these two forms:

```
phrase (comment) <local@domain>
local@domain (comment)
```

Although the algorithm for generating such a rendering is implementation specific, the following is recommended.

1. if a phrase is present, return that as the user-friendly rendering; otherwise,

2. if at least one comment is present, take the first one, remove the parenthesis, and return that as the user-friendly rendering; otherwise,

3. if the local-part does not appears to be in the syntax defined by RFC 1327 (e.g., a collection of /key=value/ strings), then return the local-part as the user-friendly rendering; otherwise,

4. if a string of the form

   /PN=value/

   is present in the local-part, then replace any dots in "value" with spaces and return that as the user-friendly rendering; otherwise,

5. if a string of the form

   /S=value/

   is not present, then return the local-part as the user-friendly rendering; otherwise,

6. if a string of the form

   /G=value/

   is present , then return "G-value S-value" as the user-friendly rendering; otherwise,

7. return "S-value" as the user-friendly rendering.

Major Changes from previous draft

NEW STUFF: SafeTcl_Services, SafeTcl_loadlibrary, SafeTcl_genid, SafeTcl_setconfigdata, MIME_ConfirmAction, "all" property for SafeTcl_getbodyprop. Added SafeTcl_{sendmessage,printtext,savemessage}, the restricted interpreter versions of their MIME_* counterparts. Added description of ~/.safetclrc for user init file.

CHANGES: untrusted_eval -> declareharmless. toplevel -> mkwindow. Implementation restrictions imposed on destroy, pack, place. eval_in_safetcl -> restrictedeval. Changed parameter order for SafeTcl_getbodyprop. Changed the successful return value from 0 to "" for several functions.

DELETED: SafeTcl_encryptstring.