THE COSY INFINITY GRAPHICAL USER INTERFACE SUBSYSTEM
MSU HEP REPORT #111101

ALEXANDER WITTIG, MARTIN BERZ, AND KYOKO MAKINO

Abstract. In this report, an extension of COSY INFINITY is described that allows output to be sent to a graphical user interface (GUI) instead of a traditional text-based console. A GUI allows the development of user friendly interfaces that can also be used by people unfamiliar with the COSY INFINITY programming environment, as well as experienced users.

We discuss in detail the design of both the application programming interface (API) for COSY programmers, as well as the internal implementation of the GUI within the COSY INFINITY source code.

1. Motivation

While COSY INFINITY provides various powerful numerical tools, their proper use typically requires expert knowledge of the programming environment. It is possible to write a user interface in the console based text mode of COSY, however, these interfaces tend to be not very user friendly. Especially if long values have to be entered, or many values are input repeatedly, a console based interface quickly reaches its limits.

To provide the power of COSY INFINITY to non-expert users, we have designed an application programming interface (API) from within COSYScript which allows the developer of an application to easily and quickly provide the user with a powerful, platform independent graphical user interface to interact with the application.

We believe this effort is of great importance in fostering the application of the tools developed by theorists to practical applications. The algorithms and methods developed to simulate various physical systems are often very complex and are developed by experts in the field. However, often these simulations cannot be run efficiently by the users for whom they were made because of the steep learning curve involved.

Having the ability to provide simple point and click interfaces to change parameters of a simulation makes the underlying algorithms much more useful to the users. This hopefully prevents powerful computational tools from being underused due to hurdles in the user interface.

2. Application Programming Interface

The programming of GUI interfaces from within COSYScript fits in naturally with the classic way of handling input and output in this environment, while providing a wide range of commonly used GUI elements, known from other programs.

Date: November 1, 2011.

Figure 1 shows a screen shot of an example showcasing all available GUI elements on the Mac OS X platform.

To define the GUIs, COSY provides the special GUI unit numbers -201 ...-210, each of which represents one window in the user's graphical environment. COSY programs define and read the content of a GUI window using standard `READ` and `WRITE` calls.

Existing programs can be very easily converted to use a GUI with only minimal modifications to existing code (see Section 2.1). This method provides a fast and efficient way to use the new GUI features in existing projects. For more sophisticated GUIs, a variety of special GUI commands can be written to the GUI unit numbers to define the elements in each window and to interact with them (see Section 2.2).

The GUI also allows input from and output to the traditional console units 5 and 6 in a GUI program. These calls are automatically routed to a separate terminal window if COSY is run in a GUI environment. That means that the same COSY program that runs in the console version will also run with a console in the GUI version of COSY (see Figure X).

Similarly, a simple ASCII based GUI is shown instead of a GUI window when a COSY program using the GUI is run in a non-GUI environment (e.g. directly from the command line or via SSH). This allows "graceful degradation" in case the optimal GUI system is not available without rendering the program useless.

2.1. **GUI Basics.** The main conceptual difference between a GUI and the traditional console based I/O is the concept of a *delayed read*. In a traditional console based user interface the program prompts the user for each value in the order specified in the program, which are read one after the other. In a GUI window, on the other hand, the user can enter values into various fields and modify them in any order before pushing a button, which then causes all values to be read in at once.

This concept is integrated into COSY by making READ commands to the GUI window units delayed. That means that COSY will not immediately read a value and place it into the variable passed to the READ command. Instead, COSY will associate each variable with a GUI input field, and only place values in the variables once a delayed read is initiated. At what point in the code such a delayed read is to be performed, is up to the programmer to specify.

2.1.1. *Simple GUI.* In order to write a simple GUI interface, or to convert an existing program using traditional console based I/O to a GUI program, a developer generally has to perform the following steps:

Instead of the usual `WRITE 6` statements to write output to the console, `WRITE -201` statements must be used to output to a GUI window instead. Similarly, `READ 5` statements are replaced by READ -201 to read input from a GUI window instead of the terminal. Note that the GUI unit number is the same for both `READ` and `WRITE` commands.

To initiate the delayed read, the procedure call `GUIIO -201;` must be initiated at the correct places in the code. This command will automatically add an OK button at the end of the window, show it to the user, wait for the button to be pushed, and then fill in the values from each input field into the variables specified in the preceding `READ` calls. The window content is discarded after this call.

The `WRITE` commands to the GUI unit will output each string it is passed as a line of simple text ("label") in the GUI, while all other data types will appear

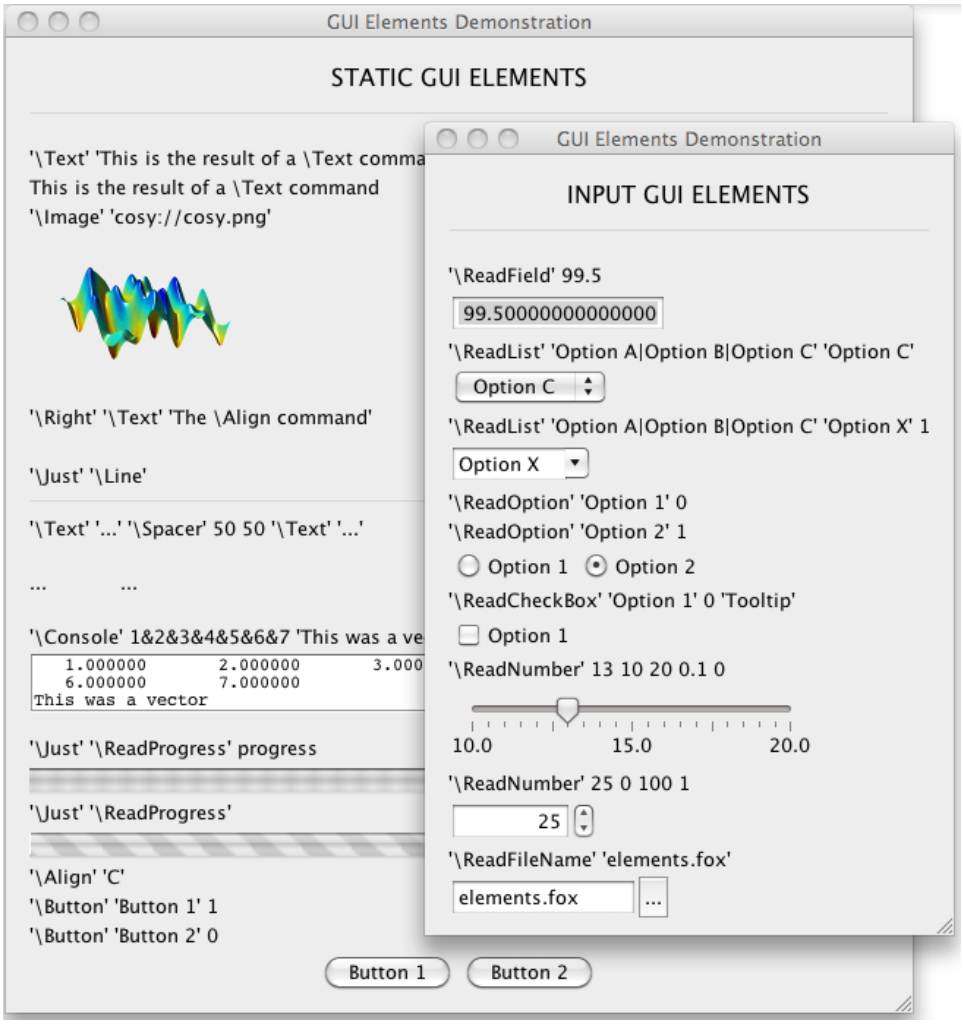

FIGURE 1. Demonstration of available GUI elements in the COSY GUI on Mac OS X.

in an embedded console in the GUI window the same way they would appear in a terminal. The user can select and copy content out of an embedded console, and scroll if the content is too long. Consecutive output into a console is appended to an existing console until a string is written to the window, creating a label after the embedded console.

For each `READ` from a GUI unit, COSY will insert an input field in a separate line in the GUI. The variable to be read is associated with this input field, and its value is placed in the variable once the window is shown to the user by calling GUIIO. When converting programs to use the GUI, developers must make sure that their code is ready for the delayed read concept. In particular, the variable being read cannot be used anywhere in the code before the call to GUIIO. Furthermore, all READ commands must read into different variables to be useful, otherwise the variable will only contain the value of the last READ command.

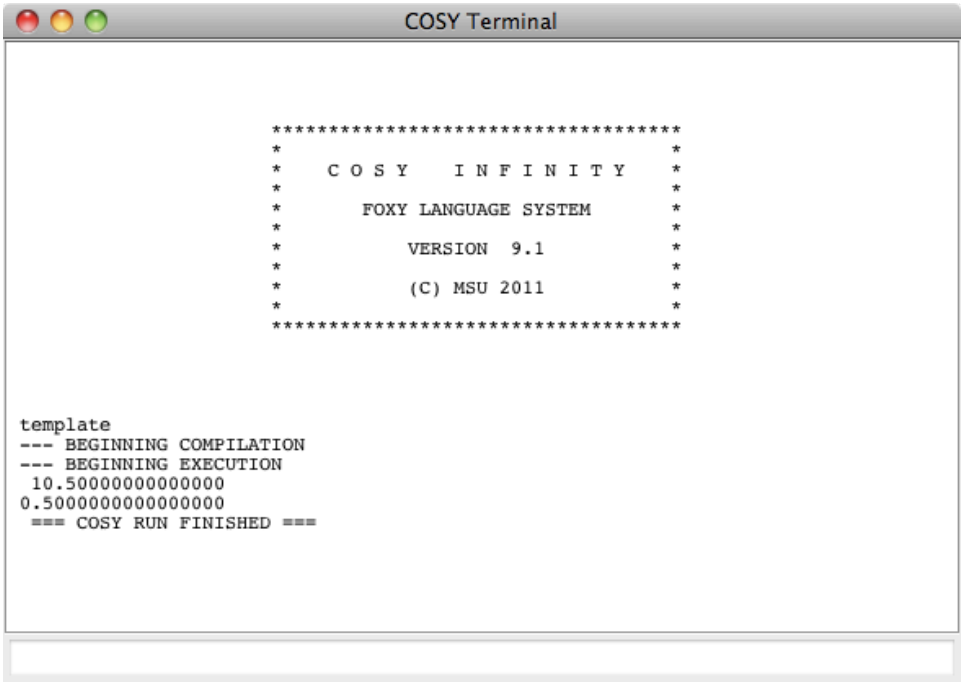

FIGURE 2. The GUI console on the Mac OS X operating system.

2.2. **Advanced GUI.** For more fine grained control over the appearance of the GUI, the full GUI interface can be controlled through special GUI commands written to the GUI window units. The COSY GUI operates with double buffered windows, that is for each window number there is the currently displayed window and a second hidden window. GUI commands defining GUI elements always act on the hidden window, while some other GUI commands can be issued to manipulate the currently displayed window (if any).

In general, the code structure to define a GUI window looks very much like the traditional console based I/O code, where the user is prompted for some input through a `WRITE` and the input is then read form the user by a `READ`. In COSY's GUI model, the GUI window is still constructed by issuing `WRITE` commands to prompt the user for input, immediately followed by `READ` commands to read back the actual input. The `READ`s are automatically delayed by COSY until a delayed read is initiated (see below).

GUI commands are issued by writing to the corresponding GUI window output unit using `WRITE`. GUI commands are strings starting with the backslash character (\), e.g. \ReadField, mimicking LaTeX notation. Each GUI command can take a number of arguments. Those are specified as additional arguments to the WRITE call. Their type can be anything COSY can convert into a string using the ST function. A single WRITE command may contain several GUI commands one after the other.

To read back a value from a GUI element, a `READ` command is issued to the GUI window unit. This associates the variable given to the `READ` command with the most recently written GUI field that can return a value, provided it has not been

associated yet. If there is no such field, either because no GUI field that returns a value has been written yet or because the last GUI field has been associated ("read") already, the `READ` command will instead insert a new text input field (via \ReadField) on its own line into the GUI window, and associate the variable with that field.

To initiate the delayed read into the thusly associated variables, the command `GUIIO` is used. It can be used in two different ways, depending on how it is called:

`GUIIO` `<unit>` `;` If called with only one argument, <unit> specifies the GUI window unit to read from. The command adds an OK button at the end of the window if no button was defined yet, shows the window, waits for a button to be pushed, reads all values from the window, and then closes the window.

`GUIIO` `<unit>` `<button>` `;` If called with two arguments, <unit> specifies the GUI window unit, and <button> must be a variable to receive the text on the button that was pushed. In this more advanced form, `GUIIO` only waits for a button to be pushed in the currently displayed window, and then reads the values of all associated variables. The text on the button that was pushed is stored in <button> (note that this string is subject to COSY's usual `READ` processing). It does not modify the window in any way (e.g. showing it, adding buttons, or closing it). If there is no window currently displayed, all variables are filled with zeros immediately and the number -1 is returned in <button>. If there is a window displayed, but it does not have a button, all variables are read immediately and the number 0 is returned in <button>.

2.2.1. *GUI Layout.* Components in the GUI are arranged based on the order in which they are added and the natural size for each element as determined by the GUI program. Each element is added at the end of the current line in the GUI. A line can be ended using the \NewLine GUI command, which causes all further elements to be added at the beginning of the next line.

The size of the window is determined by the width of the longest line, and the total number of lines. If a line is shorter than the width of the resulting window, it is aligned according to the alignment specified by one of the GUI commands \Left, \Center, \Right, or \Just. \Just will cause the elements in the line to be resized such that they fill up the entire line. By default, if none of the alignment commands was issued, lines are left justified.

Alignment commands can be called at any time, before or after writing elements to a line. It always applies to the current line and if called multiple times within the same line, the last call carries.

For more sophisticated layouts, the COSY GUI specification supports the \NewCell GUI command. With this command, it is possible to lay out elements in a tabular grid, where each cell behaves much like the lines described above. Each cell can have its own alignment, and the size of each row and column in the table is determined by the largest cell in the row or column. A row of cells is ended by calling the \NewLine GUI command.

By providing an integer argument to \NewCell, the current cell can be made to span multiple cells. The last cell in each row is automatically expanded to the end of the window, so it is not necessary to provide a cell span for the last cell. If this

behavior is not desired, an empty cell can be inserted after the last occupied cell by simply calling \NewCell.

2.2.2. *Passthrough Unit.* The special unit number -999 can be used for both reading and writing. This unit represents the *passthrough unit*, which passes input and output without modification to the standard input and output of the GUI program if COSY is running with a GUI program attached (otherwise it is equivalent to units 5/6).

That way, it is possible to connect the GUI program's input and output streams to yet another program which in this way can communicate with the COSY program.

A `READ` command from unit -999 will read a line from the GUI program's standard input. A WRITE command to unit -999 writes its arguments to the GUI program's standard output.

2.3. **GUI Command Reference.** Table 1 lists all available GUI commands currently implemented in COSY. These commands are issued by writing them to a GUI unit as described in the previous section. The first column gives the name of the command and any arguments to the command, if applicable. Commands are case insensitive, the spelling used here is by convention but not required. Commands starting with "`Read`" insert a GUI element that can be read by a subsequent `READ` call. Optional arguments are indicated by a default value in parenthesis, if they are omitted, this value is used. Optional arguments can only be omitted beginning with the last argument.The second column specifies which of the two windows at the given unit the command acts on (either the hidden or currently displayed one). The third column indicates whether a command returns a value when a delayed read is executed using the `GUIIO` call.
In the following we give some further remarks on specific GUI commands:

`\Console:`
: All arguments are output in the same form as on a regular terminal. An embedded console is inserted into the GUI window on a separate line. Output is appended to this console until another GUI component is added, or one of `\NewLine`, `\NewCell`, or `\Show` are called. The user can select and copy text in an embedded console, scroll if the text is too long, but cannot change the content.

`\Image:`
: Image file names are specified with forward slashes (/) as path separators or as `file:///` URLs for full paths. Any fully qualified URL can be given to load images over the internet (if the computer has an internet connection). The Java GUI shipped with COSY INFINITY comes with some commonly used icons built in which can be accessed using URLs of the form `cosy://yes.png`, where instead of "yes.png" any one of the built in icons ("ask.png", "clock.png", "cosy.png", "info.png", "msu.png", "msupa.png", "no.png", "star.png", "warn.png", "wrench.png", "yes.png") can be used.

`\ReadNumber:`
: When editable, this will display an input field with adjoining up and down buttons. Only numeric input is allowed in this field. When not editable, a slider is shown which can be dragged by the user to indicate a numeric value. When read, this field always returns a number in COSY.

`\ReadOption:`
Options are a group of GUI elements of which only one can be selected at a time (typically displayed as round buttons). In order to designate which option belongs to which group, the name of an option group can be specified. Of all options in a group with the same name, at most one is selected at each time.

`\ReadList:`
Presents the user with a list of options from which to select one. If the list is set to editable, the user is allowed to enter a value that is not on the list, otherwise the user must select a value on the list.

`\ReadProgress:`
This element can either display a progress bar with the given percentage of completion, or a bar with an indeterminate state to indicate that a computation is ongoing but the total time is not known. When read, this element will simply return the value it is currently set to. This is mostly so that it can have its value changed while the window is displayed using `\Set`.

`\NewLine,:` `\NewCell`, `\Left`, `\Center`, `\Right`, `\Just`
See Section 2.2.1 for information about the layout of GUI elements.

`\Deactivate,:` `\Activate`
These commands make the currently displayed window either inactive or active. In an inactive window, the user cannot interact with the elements of the window any more, they are often shown in gray. This command does not change window visibility, the window remains visible all the time. By default, windows are active, i.e. the user can interact with the elements in the window.

`\Show:`
This command closes and destroys the currently displayed window, if any, and replaces it by the hidden window, which is made visible to the user. If no arguments are given, the new window is shown where the previously displayed window was, otherwise it is centered on the screen. If both coordinates are given, the window's top left corner is positioned accordingly, with $(0, 0)$ being the top left corner of the screen, and $(1, 1)$ the bottom right.
All subsequent calls to create GUI elements will act on a newly created, initially empty hidden window. This command returns immediately, to wait for user input use `GUIIO`.

`\Set:`
To update the value of a component in the currently displayed window, this command can be called. The number of the element is determined by the order in which GUI elements were added to the window counting only elements that return a value starting with 1.

`\Debug:`
Set the debug level for the GUI. Integer between 0 and 3, with 0 (the default) meaning no debug output, 1 meaning errors are logged to the Terminal, 2 also outputting diagnostic messages, and 3 echoing the entire GUI protocol read from COSY. Can be called several times to turn debugging of certain parts of the GUI on or off.

Command & Arguments	**Window**	**Value**
\Console: Write to embedded console Any number of arguments of any type	hidden	No
\Text: Static text String to be inserted	hidden	No
\Image: Static image Image filename	hidden	No
\Line: Vertical line	hidden	No
\Spacer: Transparent element Width in pixels Height in pixels (0)	hidden	No
\Button: Push button Text on button 1 - default button, 0 - otherwise (0) Tooltip (none)	hidden	No
\ReadCheckbox: Checkbox Text next to checkbox (none) 1 - selected, 0 - not selected (0) Tooltip (none)	hidden	Yes
\ReadOption: Radio button Text next to radio button (none) 1 - selected, 0 - not selected (0) name of button group (none) Tooltip (none)	hidden	Yes
\ReadField: Unformatted input field Initial value (none) Tooltip (none)	hidden	Yes
\ReadNumber: Numerical input Current value Minimum value Maximum value Increment ((Max-Min)/100) 1 - editable, 0 - not editable (1) Tooltip (none)	hidden	Yes
\ReadList: Selection from list List of entries separated by | Initially selected value 1 - editable, 0 - not editable (0) Tooltip (none)	hidden	Yes
\ReadFileName: File selector Initial value (none)	hidden	Yes
\ReadProgress: Progress bar Progress in % or -1 (-1)	hidden	Yes

TABLE 1. Available GUI commands in COSY INFINITY.

Table 1 (continued)

Command & Arguments	**Window**	**Value**
\NewLine: Jump to next line	hidden	No
\NewCell: Jump to next cell Width of cell (1)	hidden	No
\Left: Set current cell's alignment to left	hidden	No
\Center: Set current cell's alignment to center	hidden	No
\Right: Set current cell's alignment to right	hidden	No
\Just: Set current cell's alignment to justified	hidden	No
\Title: Set window title Window title	hidden	No
\Deactivate: Make non-interactive	displayed	N/A
\Activate: Make interactive	displayed	N/A
\Show: Display window x coordinate (center) y coordinate (center)	hidden	N/A
\Close: Close and destroy window	displayed	N/A
\Set: Set value of interactive element Number of element Value to be set	displayed	N/A
\Focus: Make this the active window	displayed	N/A
\Debug: Set GUI debug level Debug level between 0 – 3	all	N/A
\Finish: Add a button if none is there yet Text on the button ('OK')	hidden	No

2.3.1. *Examples.* The following examples illustrate the use of these commands in a COSY program:

- `WRITE -201 '\NewLine' '\NewLine' '\NewLine';`
  Inserts three empty lines in window number -201.
- `WRITE -201 '\ReadNumber' tax 0 100;`
  Inserts a slider with its initial value taken from variable tax and minimum value 0 and maximum value 100 in window number -201.
- `WRITE -201 '\ReadField' name '\NewLine';`
  Inserts an input box with initial text taken from variable name followed by a new line in window number -201.
- `WRITE -208 '\ReadList' 'Ra|Zeus|Jupiter|Other' 'Zeus' 0 'Select yours!';`
  Inserts a non-editable list with the options "Ra", "Zeus", "Jupiter", and "Other", with "Zeus" initially selected and a tooltip of "Select yours!" in window number -208.
- `WRITE -201 '\Show'; GUII0 -201 button;`
  Show window number -201 and wait for a button to be pushed in this window. The name of the button is stored in variable button.

A more complete set of example programs for the use of the COSY GUI is distributed with COSY INFINITY. A demonstration of a full GUI program is given in the program gui.fox (also included in the Appendix of this report). An overview over all available GUI elements and what they look like is given in the program elements.fox. For an example of how to covert an existing program into a GUI program, see the brief COSY demo in the program briefdemo.fox, or the COSY beam physics demo in demo.fox.

3. Implementation

On the COSY internal side, the GUI interface is implemented as a separate program. The main COSY process communicates with this program through its standard input and output channels using a special ASCII based GUI protocol (see Section 3.2). The GUI program then interprets the output based on this protocol and assembles the GUI windows, manages user interaction, and handles their display.

The GUI program also executes the COSY command line executable after the user has selected the program to run. From a user's point of view, the GUI program is the single point of interaction with COSY. The distinction between the actual COSY executable that performs the actual computations and the GUI program is fully transparent to the user.

This separation of the actual COSY executable and a controlling GUI program has several advantages. The COSY source code itself remains simple and without dependencies on external GUI libraries. There are no platform dependent parts in the COSY executable, since all communication is done by writing to the FORTRAN default units 6 and 7. This allows the compilation of the very same COSY source code on a wide range of machines, with or without a graphical user interface, including high performance computing clusters which are typically only available via SSH.

Furthermore, this approach allows various GUI front ends to be developed for different purposes. We have implemented a reference implementation of a GUI program in Java which we call the Java GUI. This program is shipped with the

COSY INFINITY installer packages. However, if for some reason a different GUI interface is needed, it can be swapped out transparently as long as the new interface adheres to the same GUI protocol as described in Section 3.2.

Lastly, since the communication between the actual COSY process and the GUI program happens through an ASCII based protocol, it is easy to redirect the input and output streams to different machines e.g. using SSH. This allows the execution of COSY on a remote machine while displaying the GUI locally, much like tunneled X11 sessions.

3.1. **Java GUI.** We chose Java[**?**] as the language to implement our reference GUI program in. The main advantage of Java is its platform independence and relatively wide reach. By going this route, the same GUI program is available on Windows, Mac OS X and Linux platforms without any changes or needs to recompile the program.

Our implementation supports the full extend of all features described in the GUI protocol, and it is available in the default COSY INFINITY installer packages. In order to run the Java GUI program distributed with the COSY INFINITY installer packages for Windows and Mac OS X, at least Java 5 is required.

A typical user will run the Java GUI through the file association set up by the installer package. In Windows and Mac OS X, any FOX file should provide an entry "Run COSY" (Windows) or "Open With → Run COSY" (Mac OS X) in its context menu. Alternatively, the GUI can be started by selecting "Run COSY" from the Start Menu (Windows) or Applications folder (Mac OS X). The Java GUI will then prompt the user to navigate to a FOX file to run.

For Unix users and advanced users, as well as GUI debugging, the GUI can also be run manually from the command line. The basic command line to start the GUI this way is `java -jar COSYGUI.jar <options> <foxfile>` where <options> indicates one or more of the command line options (see below) and <foxfile> is the name of the FOX file to run. If no file is specified, the GUI will prompt the user for a file at startup.

The following options are available:

- **-d:** Can be specified up to 3 times, each one provides more debugging output on the Java console.
- **-C:** If specified, <foxfile> is not interpreted as a file name, but as a command to run that will provide an interface to a COSY process. Can be used to run COSY on a remote machine via e.g. ssh. If no command is provided, the GUI will prompt the user for a command.

The Java GUI program tries to locate the COSY executable to execute by searching the following locations for "cosy.exe" (Windows) or "cosy" (Mac OS X, Unix) in order, using the first find:

(1) Location of the FOX file being run
(2) Location of the COSYGUI.jar file
(3) Operating system dependent location where COSY was installed
(4) Default operating system search path for executables

In order to use a custom built COSY executable (e.g. because of memory requirements or other special changes), one can simply copy the executable into the same directory as the FOX files being run. Then the GUI can be run by the means described above, and will automatically pick up the correct COSY executable.

3.1.1. *Remote GUI Execution.* As mentioned above, the -C option can be used to execute arbitrary commands in order to obtain access to COSY input and output streams. This is particularly useful for remove execution, where the actual COSY process runs on some remote machine, while the GUI output is displayed on the users local machine. This can be done by the following commands on Mac OS X / Unix using an SSH connection.

First, public key authentication for the remote host must be set up for the user used to connect to the remote machine. There must not be any password prompts during the login phase for the remote connection to succeed. Furthermore, the COSY executable must be available on the remote machine, as well as the COSYScript program to be executed.

A suitable SSH command used to connect to the remote server and start COSY there is e.g. ``ssh USERNAME@www.example.com cosy -gui path/to/my/program.fox''. Of course the proper path to the COSY source and executable on the target machine must be specified.

3.2. **GUI Protocol Description.** The GUI protocol describes the internal format of the ASCII data stream used to communicate between the COSY executable and the GUI program. This data stream consists of the usual COSY output interspersed with special GUI protocol commands and is read line by line. GUI commands are lines marked in a special way to indicate that they are not part of the output. The GUI command marker is <*%GUI%*> at the beginning of a line, and </*%GUI%*> at the end.

Anything between these two markers is considered to be a special *GUI protocol command* to be interpreted by the GUI program. Other lines are *regular output*, to be redirected to a window, console or the passthrough unit in some way by the GUI program depending on the output redirect mode (see REDIRECT command below).

The format of a GUI command loosely follows the format of Unix command line parsing. A GUI command is parsed into several arguments, the separation occurs at blank spaces. Arguments containing spaces can be enclosed in double quotation marks. Any character to be included verbatim in an argument can be preceded by a backslash. The character following the backslash is added to the argument as is, without any special meaning for the parsing.

The following examples show how a GUI command string is parsed into its separate arguments:
`2 text "'\\ReadFileName' 'elements.fox'"` is parsed into 3 separate arguments: `2`, `text`, and `'\ReadFileName' 'elements.fox'` (the last is one single argument).

The GUI program then inspects the GUI protocol command argument by argument. Every GUI command is preceded by a GUI unit number in the first argument. The GUI protocol command is said to be sent to this GUI unit. In the above example this GUI unit number is 2. This unit number is followed by the case insensitive name of the GUI protocol command to be executed, "text" in the above example. All following arguments are parameters to the GUI protocol command, possibly affecting its behavior depending on the GUI protocol command.

The GUI protocol numbers GUI windows from 1 through 10, with 10 being an arbitrarily set upper limit on the number of windows available. Unit number 0 represents the GUI console window, and GUI number -1 indicates the passthrough unit (see Section 2.2.2).

3.2.1. *Output Redirection.* Lines not identified as GUI commands are to be displayed either in a GUI window, the GUI console window, or passed to the passthrough unit (see Section 2.2.2). The determination of which unit to send this output to is done by the currently set output redirection unit in the GUI program. When the GUI starts, initially the output redirection unit is 0, meaning any line not identified as a GUI protocol command is written to the GUI console window.

The GUI protocol command REDIRECT is used to change the output redirection unit in the GUI to any other valid unit number. To do so, the REDIRECT command is sent to the GUI unit to change the output unit redirection to. All subsequent output will then be written to that GUI unit number. To change the output redirection unit to GUI window 5, the GUI protocol command to be issued would be `5 REDIRECT`.

Output to a GUI window is collected and added to an embedded console within the GUI window. Output to the passthrough unit is written to the standard output stream of the GUI program. Output to the GUI console is written to the GUI console window.

3.2.2. *List of GUI Protocol Commands.* Table 2 shows all GUI protocol commands currently supported by the GUI protocol. These commands are similar to the GUI API commands shown in Table 1, however note well that these GUI protocol commands do not start with a \ (backslash character). Furthermore, there are additional GUI protocol commands used to control internals of the GUI process that are not exposed to the user (such as redirection handling).

Each command is listed with its name and its arguments, if any, on the following lines. If an argument has a default value, it is given in parenthesis. Such arguments may be omitted and the default will automatically be used. Omitting required arguments without default values is an error and the GUI program will ignore such commands, possibly issuing a warning to the user.

The unit column identifies which GUI units the command can be applied to. -1 indicates the passthrough unit, 0 the GUI console, and 1+ means any of the GUI window units.

For a description of the GUI protocol commands controlling the content of the GUI windows, see the descriptions in Section 2.3. In the following, only the additional commands controlling GUI internal functionality will be described.

`DEBUG:`
: The debugging level indicates how much additional information the GUI should output. This is mostly to help developers debug their GUI code. In normal operation, the debug level is 0, meaning the GUI program will silently ignore any invalid or incomplete GUI protocol commands. Debug level 1 will issue warnings if unknown, invalid or incomplete GUI protocol commands are encountered. Debug level 2 will output additional informational messages at the digression of the GUI program programmer. Debug level 3 will echo the entire communication between the GUI program and the COSY executable. Furthermore, it may display additional borders in the GUI windows helping in debugging alignment issues. All debug output is written into the GUI console.

`REDIRECT:`
: Sets the GUI output redirection unit. See Section 3.2.1.

`READ`:

The `READ` GUI protocol command is used to read from a GUI unit. If the GUI unit is the passthrough unit (-1), or the GUI console unit (0), the `READ` GUI protocol command reads a single line from the GUI programs standard input or the GUI console input respectively and writes this line to the COSY executables standard input. The argument to the `READ` call, if any, is ignored in this case. If the `READ` GUI protocol command is issued to a GUI window unit ($1+$), the GUI program will first write the number of values it will return as an integer to the COSY executables standard input. A value of 0 is given if there will be no values returned. This is followed by exactly the given number of values, each on a separate line. If no argument is given for the `READ` call, the value of every interactive GUI element in the window is returned. If an argument is given, only the value of this specific GUI element is returned. Counting starts with 1 and is in the order in which interactive GUI elements are defined in the window. If no such element exists, or the argument is invalid, 0 is returned and no values are written to the COSY executable.

To maintain synchronization between the GUI program and the COSY executable it is crucial that always exactly the number of values (i.e. lines) is written that is announced in the very first line.

`WAIT`:

Instructs the GUI program to wait for a button to be pressed in the given window. While the GUI program is waiting, other GUI protocol commands may still be issued and the GUI program should interpret them, i.e. the GUI program should not block while waiting for a button to be pressed. Once a button is pressed, the GUI program writes the text on the button to the COSY executables standard input in a single line. If at the time of the `WAIT` GUI protocol command the window at the given unit number is not visible, or does not contain a button, an empty line or the value 0 is written to the COSY executable's standard input immediately. The GUI program should cache users clicks, so that if the user clicks a button while the GUI program is not waiting at the given window unit, the last button pressed is stored and immediately returned on a `WAIT` GUI protocol command issued at a later time.

If the WAIT command is issued to a GUI unit number not representing a GUI window, the value -1 is immediately written to the COSY executable's standard input.

Command & Arguments	**Unit**
`TEXT`: Insert static text String to be inserted	1+
`IMAGE`: Insert static image Image filename	1+
`LINE`: Insert vertical line	1+
`SPACER`: Insert transparent element Width in pixels Height in pixels (0)	1+
`BUTTON`: Insert push button Text on button 1 - default button, 0 - otherwise (0) Tooltip (none)	1+
`READCHECKBOX`: Insert checkbox Text next to checkbox (none) 1 - selected, 0 - not selected (0) Tooltip (none)	1+
`READOPTION`: Insert radio button Text next to radio button (none) 1 - selected, 0 - not selected (0) name of button group (none) Tooltip (none)	1+
`READFIELD`: Insert unformatted input field Initial value (none) Tooltip (none)	1+
`READNUMBER`: Insert numerical input Current value Minimum value Maximum value Increment ((Max-Min)/100) 1 - editable, 0 - not editable (1) Tooltip (none)	1+
`READLIST`: Insert selection from list List of entries separated by | Initially selected value 1 - editable, 0 - not editable (0) Tooltip (none)	1+
`READFILENAME`: Insert file selector Initial value (none)	1+
`READPROGRESS`: Insert progress bar Progress in % or -1 (-1)	1+

TABLE 2. Available GUI protocol commands in COSY INFINITY.

Table 2 (continued)

Command & Arguments	**Unit**
`NEWLINE` / `NL`: Jump to next line	1+
`NEWCELL` / `NC`: Jump to next cell Width of cell (1)	1+
`LEFT`: Set current cell's alignment to left	1+
`CENTER`: Set current cell's alignment to center	1+
`RIGHT`: Set current cell's alignment to right	1+
`JUST`: Set current cell's alignment to justified	1+
`TITLE`: Set window title Window title	1+
`DEACTIVATE`: Make non-interactive	1+
`ACTIVATE`: Make interactive	1+
`SHOW`: Display window x coordinate (center) y coordinate (center)	1+
`CLOSE`: Close and destroy window	1+
`SET`: Set value of interactive element Number of element Value to be set	1+
`FOCUS`: Make this the active window	1+
`FINISH`: Insert button if none defined yet Text on the button ('OK')	1+
`WAIT`: Wait for a button to be pressed	1+
`DEBUG`: Set GUI wide debug level Debug level between 0 – 3	$-1, 0, 1+$
`REDIRECT`: Set GUI output redirection unit	$-1, 0, 1+$
`READ`: Initiate read from GUI unit Element number to read (all elements)	$-1, 0, 1+$ (1+)

4. Appendix

The following code listing is a fully functional demonstration program for a COSY INFINITY graphical user interface. The same code is included in the COSY INFINITY distribution package as the file GUIDemo.fox.

```
{
   GUIDemo.fox
   ***********

   This example shows a full blown GUI program making use of all of the
   advanced GUI programming methods that are available in COSY INFINITY 9.1
}
begin;
variable i 1; variable j 1;
variable temp 100;
variable button 100;
variable people 5000;
variable drink 25;
variable who 30;
variable shot 1;
variable small 1;
variable medium 1;
variable large 1;
variable bucket 1;
variable apple 1;
variable muffin 1;
variable bagle 1;
variable yoghurt 1;
variable banana 1;
variable pay 1;
variable tax 1;
variable pic 100;

function ss s i j; substr s i j ss; endfunction; { extract substring }

{ To see the COSY error handling in the GUI uncomment one of these }
{openf 44 'does-not-exist' 'old'; { produces a FORTRAN runtime error }}
{i := TM(1); { produces a COSY runtime error }}
{quit 1; { just quits }}

```

```
{ Set default values }
drink := 'Cocoa'; who := 'Lupo'; shot := 0;
small := 0; medium := 1; large := 0; bucket := 0;
apple := 1; muffin := 1; bagle := 0; yoghurt := 0;
banana := 1; pay := 2.5; tax := 6; pic := 'coffee.png';
people := 'Totoro|Asterix|Obelix|Goofy';
button := 'Yes';

{ The main program loop }
while button='Yes';
    { Define and show the main GUI }
    write -201 '\Title' 'Alex Coffee House';
    write -201 '\Center' '\Image' 'coffee.png' '\NewLine';
    write -201 '\Center' '\Text' 'Hello and welcome to Alex
        Coffee House' '16' '\Spacer' 0 50;
    write -201 '\NewLine' '\Just' '\Line' '\NewLine';
    write -201 '\Text' 'What would you like to drink:' '\
       NewCell';
    write -201 '\Just';
    write -201 '\ReadList' 'Coffee|Tea|Cocoa|Milk|Water|
       Beer|Wine|Magic Potion' drink;
    read -201 drink;
    write -201 '\NewLine';
    write -201 '\Text' 'Who''s this drink for?' '\NewCell';
    write -201 '\Just' '\ReadList' people who 1;
    read -201 who;
    write -201 '\NewLine' '\Text' 'What size would you like
       :' '\NewCell';
    write -201 '\ReadOption' 'Shot' shot;
    read -201 shot;
    write -201 '\NewLine' '\NewCell';
    write -201 '\ReadOption' 'Small' small;
    read -201 small;
    write -201 '\NewLine' '\NewCell';
    write -201 '\ReadOption' 'Medium' medium;
    read -201 medium;
    write -201 '\NewLine' '\NewCell';
    write -201 '\ReadOption' 'Large' large;
    read -201 large;
    write -201 '\NewLine' '\NewCell';
    write -201 '\ReadOption' 'Bucket' bucket;
    read -201 bucket;
    write -201 '\NewLine' '\NewLine' '\Just' '\Line' '\
       NewLine';
    write -201 '\Text' 'Extras:' '\NewLine';
```

```
76      write -201 '\ReadCheckBox' 'Apple' apple 'A healthy
            choice!';
77      read -201 apple;
78      write -201 '\ReadCheckBox' 'Muffin' muffin 'Very tasty
            !';
79      read -201 muffin;
80      write -201 '\ReadCheckBox' 'Bagle' bagle 'A classic!';
81      read -201 bagle;
82      write -201 '\ReadCheckBox' 'Yoghurt' yoghurt 'Are you
            on a diet?';
83      read -201 yoghurt;
84      write -201 '\ReadCheckBox' 'Banana' banana 'Monkey see,
             monkey do!';
85      read -201 banana;
86      write -201 '\NewLine' '\NewLine' '\Just' '\Line' '\
            NewLine';
87      write -201 '\Text' 'How much are you willing to pay:';
88      write -201 '\NewCell' '\Just' '\Text' '$';
89      write -201 '\ReadNumber' pay 1.5 10 0.1 1;
90      read -201 pay;
91      write -201 '\NewLine' '\Text' 'Your Sales Tax (in %):';
92      write -201 '\NewCell' '\Just';
93      write -201 '\ReadNumber' tax 0 100 1 0;
94      read -201 tax;
95      write -201 '\NewLine' '\Text' 'Your favourite picture
            :';
96      write -201 '\NewCell' '\Just';
97      write -201 '\ReadFileName' pic;
98      read -201 pic;
99      write -201 '\NewLine' '\NewLine' '\Center';
100     write -201 '\Button' 'Errr, no.' '\NewCell' '\Center';
101     write -201 '\Button' 'Place order!' 1;
102     write -201 '\Show';
103
104         { wait for button and read from GUI }
105     GUIIO -201 button;
106     if button#'Place order!'; quit 0; endif;
107
108     { Write the results to next window }
109     people := people&'|'&who; { append input to list of
            people for next time }
110     write -201 '\Title' 'Alex Coffee House: Review Order';
111     write -201 '\Text' 'Thank you for your order:' '\
            NewLine';
112     write -201 '\Console' 'Here is your order for '&who '';
```

```
113     if drink='Coffe'; write -201 '\Console' 'Our most
           awesome Coffe';
114     elseif drink='Tea'; write -201 '\Console' 'Splendid,
           old chap! A nice cup of tea';
115     elseif drink='Cocoa'; write -201 '\Console' 'Yummy hot
           cocoa';
116     elseif drink='Milk'; write -201 '\Console' 'A glass of
           fresh milk';
117     elseif drink='Water'; write -201 '\Console' 'Just water
           ';
118     elseif drink='Beer'; write -201 '\Console' 'Fresh
           German beer';
119     elseif drink='Wine'; write -201 '\Console' 'Wine it is
           ';
120     elseif drink='Magic Potion';
121           if who='Obelix';
122           write -201 '\Console' 'No Obelix, you know you
                 can''t have magic potion!';
123        elseif lo(1);
124           write -201 '\Console' 'Getafix'' Magic Potion';
125        endif;
126     endif;
127     if shot=1; temp:='Shot';
128     elseif small=1; temp:='Small';
129     elseif medium=1; temp:='Medium';
130     elseif large=1; temp:='Large';
131     elseif bucket=1; temp:='Bucket';
132     endif;
133     write -201 '\Console' 'Size of your beverage: '&temp;
134     temp := '';
135     if apple=1; temp := temp&'apple, '; endif;
136     if muffin=1; temp := temp&'muffin, '; endif;
137     if bagle=1; temp := temp&'bagle, '; endif;
138     if yoghurt=1; temp := temp&'yoghurt, '; endif;
139     if banana=1; temp := temp&'banana, '; endif;
140     write -201 '\Console' 'On the side, you will get the
           following: '&ss(temp,1,length(temp)-2) '' '';
141     write -201 '\Console' 1&2&3&4&5&6&7&8 'This vector was
           a test of the vector emergency broadcasting network
           .';
142     write -201 '\NewLine' '\Center' '\Image' pic '\NewLine
           ';
143     write -201 '\NewLine' '\NewLine' '\Text' 'Please stand
           by while we freshly prepare your order!';
144     write -201 '\NewLine' '\Just' '\ReadProgress' '0';
```

```
145     write -201 '\Show'; { implicitly closes the previous
            window }
146
147         { waste some time, like in a real restaurant }
148     temp := 1000;
149     j := 0;
150     loop i 1 30000000;
151         temp := sqr(sqrt(log(exp(temp))));
152         if j#int(100*i/30000000);
153             j := int(100*i/30000000);
154             write -201 '\Set' 1 j; { set value of the
                    progress bar }
155         endif;
156     endloop;
157
158         { Present the bill }
159     RECST pay*(1+tax/100) '(F5.2)' temp;
160     write -202 '\Text' 'Your order is ready now. This will
            be $'&temp&'.';
161     write -202 '\NewLine' '\Text' 'Thanks for stopping by!'
            '\NewLine' '\NewLine';
162         write -202 '\Image' 'cosy://ask.png';
163     write -202 '\Text' '  Would you like to place another
            order?';
164     write -202 '\NewLine' '\Center' '\Button' 'Yes' 1 '\
            Button' 'No';
165         write -202 '\Show';
166     write -201 '\Deactivate';
167         { wait for button }
168     GUIIO -202 button;
169     write -202 '\Hide';
170     write -201 '\Hide';
171 endwhile;
172
173 end;
```