# Biksel User Manual

For Biksel version 1.0.



This document is part of Biksel, published by Fivey Software. See https://fiveysoftware.com/biksel/ for more information.



## Contents

# Introduction to Biksel

Biksel is a programming system designed for making pixel art oriented video games and other applications. It is intended to be accessible to beginners with no prior programming experience, and is loosely inspired by the BASIC systems of old. Biksel programs are written in the C programming language, or more specifically a “safe” subset of C which excludes certain features such as pointers. See C Programming Language — Quick Reference for details.

Please note that Biksel is currently a work in progress: additional features and improvements are planned for the future. For more information or to report a bug see https://fiveysoftware.com/biksel/.

## Getting started

In order to write programs with Biksel you need two things:

1. A working installation of Biksel.
2. A text editor.

### Checking if you have the software installed

To test if you have a working installation of Biksel, enter the following command at the terminal:

```
bikinfo version
```

If Biksel is installed, this command will output the version number of your Biksel installation, e.g. `1.0.0`. If Biksel is not installed, you will get an error message saying “`command not found`”.

If you need to install Biksel, you can obtain a copy from the Biksel website at https://fiveysoftware.com/biksel/.

The recommended editor for writing Biksel programs is a text editor called GNU Emacs. To test if you have GNU Emacs installed, enter the following command at the terminal:

```
emacs --version
```

Similarly to before: if Emacs is installed then this command will output a version number; if not you will get an error message.

If you need to install Emacs, the easiest way to do so is usually via your operating system's package manager. For example, on a Debian-based version of GNU/Linux (including the Raspberry Pi's “Raspbian” operating system) you can install Emacs by entering the following command at the terminal:

```
sudo apt-get install emacs
```

## Setting up the Emacs text editor

There is one remaining step of preparation: you need to configure the GNU Emacs text editor so that it knows about Biksel. This will make available some useful Biksel-specific shortcut keys within the Emacs editor. If you are new to Emacs, the simplest way to do this configuration step is to enter the following command at the terminal, which will create a configuration file for Emacs in your home directory called “.emacs”:

****WARNING**: this command will overwrite any existing “.emacs” file, so if you already have such a file be sure to make a backup of it first.**

```
cp $(bikpath emacs-init-file) ~/.emacs
```

(Note: the “`$(…)`” syntax in the above command is a feature of the Bash shell language known as “command substitution”: it passes the output of the `bikpath` command as the first argument to the `cp` command; the `bikpath` command outputs a path to a template Emacs configuration file provided by Biksel; the `cp` command copies a file; the “~” symbol refers to your home directory.)

For more information about the `bikpath` command, see Biksel Shell Commands — Quick Reference.

If you are already an Emacs user and have your own “.emacs” init file, please see the appendix at the end of this introduction instead of using the above command.

# Using Biksel

## Writing a program

The typical workflow for writing a Biksel program is as follows. Using your preferred file manager, create a new directory for your program and give it a name ending in “`.bik`”. For example, “`hello-world.bik`”. Next create an empty file inside this directory and give it the name “`main.c`”. This file is going to contain the C source code for your program.

Open the GNU Emacs text editor. You can do this either by finding it in your desktop environment's applications menu, or by entering the following command at the terminal:

```
emacs
```

To start editing your “`main.c`” file in Emacs, drag and drop the file from your file manager into the Emacs text editor window. Then, using Emacs, add some code for your program to “`main.c`”. For example, the following

minimal program will display the message “Hello, world!” in a window, wait for the user to press a key, then exit:

```
void Main(Channel io)
{
  BeginDraw(io);
  BeginDrawText(io, Mono11, White, 10, 230);
  PutText(io, "Hello, world!");
  EndDrawText(io);
  EndDraw(io);

  WaitForKeyPress(io);
}
```

For more information on using the Emacs text editor see GNU Emacs Text Editor — Quick Reference. For more information on writing Biksel programs see C Programming Language — Quick Reference and Biksel Programming Interface — Quick Reference.

## Building and running your program

To build a Biksel program, press F5 within Emacs. This will automatically save any changes that you have made to your program source file. It will also open up a compilation output subwindow in Emacs to tell you whether or not the build has succeeded. If the build has failed (e.g. due to a typo in your program), you will see error messages highlighted in red: you can click on an error message and it will act as a hyperlink to the part of your program that triggered the error.

Once your build has succeeded, press F6 within Emacs to run your program. Note that the compiled executable will have been placed in an automatically-generated subdirectory called `build-output` within your program's directory.

For convenience it is also possible to build and then immediately run a Biksel program in a single step by pressing F7 within Emacs.

## Editing images

To open the Biksel image editor, press F8 within Emacs. For information on using the image editor see Biksel Image Editor — Quick Reference.

When saving an image to disk from the Biksel image editor it is conventional to use a file name ending in “`.sprite`”. Such image files have a format called “`biksel-sprite`” and may be loaded into Biksel programs using a C procedure called “`LoadNewSprite`”. See the section “Sprite file” of Biksel File Formats — Quick Reference and the section on “Sprites” of Biksel Programming Interface — Quick Reference for details.

# Running the example programs

Biksel comes with several example programs. You can print the path to the directory containing these example programs by entering the following command at the terminal:

```
bikpath examples
```

Before running the example programs, it is recommended to make a working copy of them, e.g. in your home directory, so that you can make changes to the copy without losing the originals. For example, enter the following command to make a copy of the Biksel example programs in a new directory called “`biksel-examples`”:

```
cp -r $(bikpath examples) biksel-examples
```

(Note: the “`-r`” argument in the above command indicates a “recursive” copy, i.e. to copy a directory and all of its contents.)

To run an example program, navigate into its subdirectory within your working copy of the examples. Drag and drop the program's “`main.c`” file into Emacs. You can then build and run the program in the usual way as described above in the section on “Building and running your program”.

# Accessing this user manual offline

Copies of this user manual in HTML and PDF formats are included with Biksel. You can print the path to the included manual in your preferred format by entering one of the following commands at the terminal:

```
bikpath manual-html
```

or:

```
bikpath manual-pdf
```

Alternatively, you can instruct a program of your choice to open the manual directly with one of the following commands:

*your-web-browser* `$(bikpath manual-html)`

or:

*your-pdf-viewer* `$(bikpath manual-pdf)`

Just replace *your-web-browser* or *your-pdf-viewer* with the name of your preferred web browser or PDF viewer program.

# Appendix: information for Emacs users

This appendix describes how to configure GNU Emacs to work with Biksel if you are already an Emacs user and have your own “.emacs” init file.

You will need to add a few lines to your “.emacs” in order to enable Biksel-specific shortcut keys within Emacs. The relevant lines can be found in the template Emacs init file included with Biksel. Enter the following command at the terminal to open this template init file in Emacs:

```
emacs $(bikpath emacs-init-file)
```

The lines you will need are those beneath the heading “SECTION 1: BIKSEL”. Add them into your own “.emacs” file, making changes to the key bindings if you wish. (There is no necessity to use anything from the other sections of the template init file, although you may wish to do so if the configuration tweaks there look useful to you.)

For more information about the Biksel-specific part of the template Emacs init file, see Biksel Emacs Lisp Interface — Quick Reference.

# GNU Emacs Text Editor — Quick Reference

To open the GNU Emacs text editor, either launch it from your desktop environment's applications menu, or enter the shell command “`emacs`” at the terminal.

## Unusual aspects

### Clicking

When placing the cursor with the mouse, you click in the *centre* of a character, not in between characters.

### Selecting

When selecting a region of text, you don't click and drag. Instead, click with the left mouse button on one character then click with the right mouse button on another.

### Prompt

Take note of the single blank line at the very bottom of the editor window: this is where commands will prompt you for input and give you feedback. You can use Ctrl + G to cancel a command if it is prompting you for input.

## Commands

Note: commands labelled (*) are extensions provided by Biksel.

### General

| | |
|---|---|
| Ctrl + X, then Ctrl + C: | quit |
| Ctrl + G: | cancel command |

### File

| | |
|---|---|
| drag and drop: | open file |
| Ctrl + X, then Ctrl + F: | create or open File |
| Ctrl + X, then Ctrl + S: | Save |
| Ctrl + X, then S: | Save all |

## Edit

| | |
|---|---|
| Ctrl + /: | undo |

| | |
|---|---|
| Ctrl + K: | cut lines (press multiple times) |
| Ctrl + Y: | paste lines |

| | |
|---|---|
| Left Mouse, then Right Mouse: | copy region |
| Left Mouse, then Right Mouse twice: | cut region |
| Middle Mouse: | paste region |

## View

| | |
|---|---|
| Alt + Up: | scroll Up (*) |
| Alt + Down: | scroll Down (*) |
| Ctrl + L: | centre view around cursor |

| | |
|---|---|
| Ctrl + X, then 1: | un-split window |
| Ctrl + X, then 2: | split window vertically |
| Ctrl + X, then 3: | split window horizontally |

| | |
|---|---|
| Ctrl + X, then O: | move to Other subwindow |
| Ctrl + X, then B: | switch to other open file |

## Search

| | |
|---|---|
| Ctrl + S: | Search forward |
| Ctrl + R: | search backward |

## C

| | |
|---|---|
| Tab: | re-indent line or region |
| Alt + Q: | re-wrap comment paragraph |

## Biksel

| | |
|---|---|
| F5: | build (*) |
| F6: | run (*) |
| F7: | build and run (*) |
| F8: | open image editor (*) |

# Biksel Image Editor — Quick Reference

## Tools

| | |
|---|---|
| S: | Select |
| D: | Draw |
| F: | Fill |
| X: | set offset |

## Commands

| | |
|---|---|
| Left Mouse: | use current tool |
| Right Mouse: | pick colour |
| Space + Left Mouse: | pan |
| 1 to 5: | zoom |
| C, then F: | Fill image |
| Ctrl + R: | Resize canvas |
| Ctrl + X, then L: | Load |
| Ctrl + X, then S: | Save |
| Ctrl + X, then R: | Restart |
| Ctrl + X, then Q: | Quit |

## Alternative bindings

| | |
|---|---|
| Shift + Left Mouse | → Right Mouse |
| F1 | → Ctrl + X |

# C Programming Language — Quick Reference

## Comments

```
// Single-line comment.

/* Multi-line
comment. */
```

## Commands

| | |
|---|---|
| *Procedure*(*expression*$_1$, … *expression*$_n$); | procedure call |
| *variable*++; | increment |
| *variable*--; | decrement |
| *variable* = *expression*; | set value |
| *variable* += *expression*; | add |
| *variable* -= *expression*; | subtract |
| *variable* *= *expression*; | multiply by |
| *variable* /= *expression*; | divide by |
| *variable* %= *expression*; | remainder on division by |

In the above, "*variable*" may be followed by any number of ".*field*" structure field accessors.

## Expressions

Numbers, text in double quotes, variables, constants, and enumeration tags are expressions.

The following are operator expressions, listed in decreasing order of precedence with operators of equal precedence grouped together:

| | |
|---|---|
| *Procedure*(*expression*$_1$, … *expression*$_n$) | procedure call |
| *expression*.*field* | structure field access |
| -*expression* | unary minus |
| !*expression* | logical NOT |
| (*Type*) *expression* | type cast |

| | |
|---|---|
| *expression* `*` *expression* | times |
| *expression* `/` *expression* | divide |
| *expression* `%` *expression* | remainder |
| *expression* `+` *expression* | plus |
| *expression* `-` *expression* | minus |
| *expression* `<` *expression* | less than |
| *expression* `<=` *expression* | less than or equals |
| *expression* `>` *expression* | greater than |
| *expression* `>=` *expression* | greater than or equals |
| *expression* `==` *expression* | equals |
| *expression* `!=` *expression* | not equals |
| *expression* `&&` *expression* | logical AND |
| *expression* `\|\|` *expression* | logical OR |
| *expression* `?` *expression* `:` *expression* | if-then-else |

# Control flow

## Goto

```
label:
  ...
  goto label;
```

## If

```
if (expression) {
  ...
}
else if (expression) {
  ...
}
...
else {
  ...
}
```

## Switch

```
switch (expression) {
case integer:
  ...
  break;

...

default:
  ...
  break;
}
```

The “`else if`” and “`else`” blocks in `if` are optional, as is the “`default`” case in `switch`.

Multiple “`case` *integer*`:`” labels may be attached to a single case in `switch`.

### While

```
while (expression) {
  …
}
```

### For

```
for (command1; expression; command2) {
  …
}
```

A `for` loop is similar to a `while` loop, with the added feature that $command_1$ is executed before the loop starts and $command_2$ is executed after each iteration of the loop.

### Infinite loop

```
while (True) {
  …
}
```

### Repeat *n* times

```
Int32 i;
…
for (i = 0; i < n; i++) {
  …
}
```

You can use "`break;`" to break out of a `switch`, `while`, or `for` block.

You can use "`continue;`" to skip the rest of the current iteration of a `while` or `for` loop.

## Defining variables

```
// Uninitialized variable
Type name;

// Initialized variable
Type name = expression;

// Constant
const Type name = expression;

// Block
{
  // Variables defined here exist until the end of the block.
  …
}
```

## Defining global constants

```
// Definition
const Type Name = value;

// Definition for value of structure type
const Type Name = { value1, … valuen };
```

# Defining procedures

```
// Definition
Type Name(Type1 parameter1, ... Typen parametern)
{
  ...
}
```

You can use a return type of "`void`" for a procedure that returns nothing, or "`noreturn void`" for a procedure that does not return at all.

You can use "`return` *expression*`;`" to return from a procedure with non-`void` return type, or "`return;`" to return from a procedure with `void` return type.

# Defining data types

## Alias

```
typedef Type Name;
```

## Enumeration

```
typedef Int32 Name;
enum {
  Tag1,
  ...
  Tagn
};
```

## Structure

```
typedef struct {
  Type field;
  ...
} Name;
```

# Conventions used in Biksel

### Naming

Write all names in camel case (i.e. `thisIsCamelCase`).

Use lower case for the first letter of names that have local scope (labels, variables, local constants, and structure fields) and use upper case for the first letter of names that have global scope (global constants, procedures, data types, and enumeration tags).

### Modes and formats

When you define a procedure, write a mode declaration in a comment for each parameter of type `Channel`.

When you create a memory file that will have a particular format, write a file format declaration in a comment.

# “Gotchas” to watch out for

No programming language is perfect. C has a few aspects to its design that result in common programming pitfalls. These pitfalls come in two categories: (+) situations where the C compiler accepts code that arguably it should reject, and (−) situations where the C compiler rejects code that arguably it should accept.

## (+) Equals

Beware of accidentally setting the value of a variable when you mean to test for equality:

```
// WRONG but accepted by compiler
if (x = 3)
{
  …
}
```

```
// CORRECT
if (x == 3)
{
  …
}
```

## (−) Labels

C does not allow a label to be followed immediately by a variable definition or by the closing brace of a block. To work around this you can place a semicolon after the label:

```
// Rejected by compiler
 label:
  Type variable = expression;
  …
```

```
// CORRECT
 label:;
  Type variable = expression;
  …
```

```
// Rejected by compiler
{
  …
 label:
}
```

```
// CORRECT
{
  …
 label:;
}
```

## (+) Fall through

If you forget to break at the end of a case in a `switch` block, control will fall through to the next case:

```
// Often not what you intended
switch (expression) {
…

case integer:
  …
  // no break

case integer:
…
}
```

```
// CORRECT
switch (expression) {
…

case integer:
  …
  break;

case integer:
…
}
```

## (+) Variables in switch

If you define any variables within a case in a `switch` block, it is recommended to put the code for that case in a sub-block of its own. This ensures that the variables will not remain in scope in subsequent cases, which can lead to unexpected behaviour:

```
// WRONG: variable still in
// scope after end of case
switch (expression) {
…

case integer:
  …
  Type variable = expression;
  …
  break;

case integer:
…
}
```

```
// CORRECT
switch (expression) {
…

case integer:
  {
    …
    Type variable = expression;
    …
  }
  break;

case integer:
…
}
```

# Biksel Programming Interface — Quick Reference

## Program entry point

```
// Mode (io): Main
void Main(Channel io)
{
  // Your program here.
}
```

## Basic data types

```
/* Primitive types:

Int32       (32-bit signed integer)
Int64       (64-bit signed integer)
Float64     (64-bit floating point number)

StaticText  (immutable text specified in double quotes)

*/

typedef Int32 Bool;
enum { False, True };

typedef Int32 MayTag;
enum { Nothing, Just };

typedef Int32 ListTag;
enum { Nil, Cons };

typedef struct {
  MayTag tag;
  Int32 value; // only valid if tag = Just
} MaybeInt32;

typedef struct {
  MayTag tag;
  Int64 value; // only valid if tag = Just
} MaybeInt64;

typedef struct {
  MayTag tag;
  Float64 value; // only valid if tag = Just
} MaybeFloat64;
```

# Mathematical functions and constants

```
Float64 RoundHalfUp(Float64 x);
Float64 Floor(Float64 x);

Float64 Exp(Float64 x);
Float64 Log(Float64 x);

Float64 Pow(Float64 x, Float64 y);
Float64 Sqrt(Float64 x);

// Angles are in radians.

const Float64 Tau; // a full turn
const Float64 Pi;  // a half turn

// Components of a unit vector at an angle to the x-axis.
Float64 Cos(Float64 angle);
Float64 Sin(Float64 angle);

// Angle of a vector relative to the x-axis, between -Tau/2 and
// Tau/2.
Float64 Angle(Float64 x, Float64 y);
```

# Keyboard keys

```
typedef Int32 Key;
enum {
  // CHARACTER KEYS

  // Letters
  KeyA, KeyB, KeyC, KeyD, KeyE, KeyF, KeyG, KeyH, KeyI, KeyJ,
  KeyK, KeyL, KeyM, KeyN, KeyO, KeyP, KeyQ, KeyR, KeyS, KeyT,
  KeyU, KeyV, KeyW, KeyX, KeyY, KeyZ,

  // Digits
  Key0, Key1, Key2, Key3, Key4,
  Key5, Key6, Key7, Key8, Key9,

  KeyBackquote,          KeyHyphen,              KeyEquals,
  KeyLeftSquareBracket,  KeyRightSquareBracket,  KeyBackslash,
  KeySemicolon,          KeySingleQuote,         KeyComma,
  KeyPeriod,             KeyForwardSlash,

  // British keyboard only
  KeyHash,

  // EDITING AND NAVIGATION KEYS

  KeyEscape,  KeyBackspace,  KeyTab,
  KeyReturn,  KeySpace,
```

```
  KeyInsert,  KeyDelete,  KeyHome,
  KeyEnd,     KeyPageUp,  KeyPageDown,

  // Arrow keys
  KeyLeft,  KeyRight,  KeyDown,  KeyUp,

  // NUMERIC KEYPAD

  // Digits.
  KeyNumpad0, KeyNumpad1, KeyNumpad2, KeyNumpad3, KeyNumpad4,
  KeyNumpad5, KeyNumpad6, KeyNumpad7, KeyNumpad8, KeyNumpad9,

  KeyNumpadDecimalPoint,

  KeyNumpadEnter,  KeyNumpadPlus,  KeyNumpadMinus,
  KeyNumpadTimes,  KeyNumpadDivide,

  // LOCK AND MODIFIER KEYS

  KeyCapsLock,  KeyNumLock,

  KeyLeftShift,  KeyRightShift,  KeyLeftCtrl,  KeyRightCtrl,
  KeyLeftAlt,

  // US keyboard only
  KeyRightAlt,

  // British keyboard only
  KeyAltGr,

  // FUNCTION KEYS

  KeyF1, KeyF2, KeyF3, KeyF4, KeyF5, KeyF6,
  KeyF7, KeyF8, KeyF9, KeyF10, KeyF11, KeyF12
};
```

# Mouse buttons

```
typedef Int32 MouseButton;
enum {
  MouseLeft,
  MouseRight
};
```

# Colours

```
const Color Transparent;

const Color Black;
const Color White;
const Color Red;
const Color Green;
const Color Blue;
const Color Cyan;
const Color Magenta;
const Color Yellow;

// 'red', 'green', 'blue', and 'level' must be in the range
// 0-255.
Color Rgb(Int32 red, Int32 green, Int32 blue);
Color Gray(Int32 level);
```

# Sprites

```
// Mode (io): Main
// Returns a new channel in mode 'Sprite'.
Channel LoadNewSprite(StaticText name,
                      Channel io, StaticText path);
```

# Fonts

```
// This channel is in mode 'Font'.
const Channel Mono11;
```

# Text

```
// Mode (c): PutText
void PutText(Channel c, StaticText text);
void PutInt32AsText(Channel c, Int32 x);
void PutInt64AsText(Channel c, Int64 x);
void PutFloat64AsText(Channel c, Float64 x);
```

# View transformations

```
const Trans FlipX;
const Trans FlipY;

Trans Scale(Int32 a);
Trans Rotate(Int32 quarterTurns);
Trans Translate(Int32 x, Int32 y);
Trans ClipToRect(Int32 x1, Int32 y1, Int32 x2, Int32 y2);

const Trans Identity;
Trans Compose(Trans b, Trans a);
```

# Drawing graphics

```
// Mode (io): Main -> Draw >> EndDraw >> Main
void BeginDraw(Channel io);

// Mode (io): Draw >> EndDraw -> ()
void EndDraw(Channel io);

// Mode (io): Draw
void FillFrame(Channel io, Color col);
void FillRect(Channel io, Color col,
              Float64 x1, Float64 y1, Float64 x2, Float64 y2);
void DrawPoint(Channel io, Color col, Float64 x, Float64 y);
void DrawLine(Channel io, Color col,
              Float64 x1, Float64 y1, Float64 x2, Float64 y2);
void DrawRect(Channel io, Color col,
              Float64 x1, Float64 y1, Float64 x2, Float64 y2);

// Mode (io): Draw
// Mode (sprite): Sprite
void DrawSprite(Channel io, Channel sprite,
                Float64 x, Float64 y);
void DrawSpriteTrans(Channel io, Channel sprite, Trans t,
                     Float64 x, Float64 y);

// Mode (io): Draw -> PutText >> EndDrawText >> Draw
// Mode (font): Font
void BeginDrawText(Channel io, Channel font, Color col,
                   Float64 x, Float64 y);

// Mode (io): PutText >> EndDrawText -> ()
void EndDrawText(Channel io);

// Mode (io): Draw -> Draw >> EndTransform >> Draw
void BeginTransform(Channel io, Trans t);

// Mode (io): Draw >> EndTransform -> ()
void EndTransform(Channel io);
```

# Querying the keyboard and mouse

```
// Mode (io): Main
Bool IsKeyDown(Channel io, Key key);
Bool IsMouseButtonDown(Channel io, MouseButton button);
Float64 GetMouseX(Channel io);
Float64 GetMouseY(Channel io);
```

# Simple event handling

```
// Mode (io): Main
void WaitForTick(Channel io);
Key WaitForKeyPress(Channel io);
```

# Full event handling

```
typedef Int32 EventTag;
enum {
  Tick,                              // Tick event
  KeyPress, KeyRepeat, KeyRelease,   // Key events
  MousePress, MouseRelease,          // Mouse button events
  MouseMove                          // Mouse move event
};

typedef struct {
  EventTag tag; // KeyPress, KeyRepeat, or KeyRelease
  Key key;
} KeyEvent;

typedef struct {
  EventTag tag; // MousePress or MouseRelease
  MouseButton button;
  Float64 x;
  Float64 y;
} MouseButtonEvent;

typedef struct {
  Float64 x;
  Float64 y;
} MouseMoveEvent;

// Mode (io):
//   Main -> TickEvent         (returns Tick)
//         | KeyEvent          (returns KeyPress, KeyRepeat,
//                                 or KeyRelease)
//         | MouseButtonEvent  (returns MousePress or
//                                 MouseRelease)
//         | MouseMoveEvent    (returns MouseMove)
EventTag WaitForEvent(Channel io);
```

```
// Mode (io): TickEvent -> Main
void TakeTickEvent(Channel io);

// Mode (io): KeyEvent -> Main
KeyEvent TakeKeyEvent(Channel io);

// Mode (io): MouseButtonEvent -> Main
MouseButtonEvent TakeMouseButtonEvent(Channel io);

// Mode (io): MouseMoveEvent -> Main
MouseMoveEvent TakeMouseMoveEvent(Channel io);

// Mode (io):
//   TickEvent | KeyEvent | MouseButtonEvent | MouseMoveEvent
//   -> Main
void DiscardEvent(Channel io);
```

# Memory files

```
// Returns a new channel in mode 'File'.
Channel NewFile(StaticText name);

typedef Int32 StreamMode;
enum {
  Read,
  Write
};

// Mode (c):
//   File -> Read >> Close >> File   (streamMode == Read)
//         | Write >> Close >> File  (streamMode == Write)
void Open(Channel c, StreamMode streamMode);

// Mode (c): Read >> Close | Write >> Close -> ()
void Close(Channel c);

// Mode (c): Read
Int32 ReadInt32(Channel c);
Int64 ReadInt64(Channel c);
Float64 ReadFloat64(Channel c);
Bool ReadBool(Channel c);
MayTag ReadMayTag(Channel c);
ListTag ReadListTag(Channel c);

// Mode (c): Write
void WriteInt32(Channel c, Int32 x);
void WriteInt64(Channel c, Int64 x);
void WriteFloat64(Channel c, Float64 x);
void WriteBool(Channel c, Bool x);
void WriteMayTag(Channel c, MayTag tag);
void WriteListTag(Channel c, ListTag tag);
```

```
// Mode (c): File
void Clear(Channel c);

// Mode (c1, c2): File
void Swap(Channel c1, Channel c2);
```

# Error handling

```
noreturn void Error(StaticText context, StaticText message);
```

# Debugging

```
// This channel is in mode 'PutText'.
const Channel Trace;
```

# Biksel Mode and Format Notation — Quick Reference

The notation described here is not part of the C programming language, and is not understood by the C compiler. However, it is useful for documenting Biksel programs and should be used within comments.

## Mode expressions

The following are listed in decreasing order of operator precedence with operators of equal precedence grouped together:

| | |
|---|---|
| *Name* | primitive mode |
| `()` | empty mode sequence |
| *mode*$_1$ `>>` *mode*$_2$ | *mode*$_1$ followed by *mode*$_2$ |
| *mode*$_1$ `|` *mode*$_2$ | *mode*$_1$ or *mode*$_2$ |

## Mode declarations

```
// Constraint
Mode (parameter): mode

// Constraint with transition
Mode (parameter): mode -> mode

// Constraint with conditional transition
Mode (parameter): mode -> mode  (condition)
                        | mode  (condition)
                        ...
```

In the above, *condition* may be written informally or may be a C expression. Each *condition* may depend on the arguments and return value of the procedure to which the mode declaration applies.

In all of the above, *parameter* may be replaced by a comma-separated list of parameters.

# Format expressions

The following are listed in decreasing order of operator precedence with operators of equal precedence grouped together:

| | |
|---|---|
| *Name* | named format |
| *parameter* | format template parameter |
| `()` | empty format, i.e. zero bits of data |
| `"`*Tag*`"` | enumeration tag, represented as `Int32` |
| `[`*format*`]` | equivalent to “`ListOf` *format*” |
| *Name* *format*$_1$ … *format*$_n$ | instance of format template |
| *format*$_1$ `>>` *format*$_2$ | *format*$_1$ followed by *format*$_2$ |
| *format*$_1$ `|` *format*$_2$ | *format*$_1$ or *format*$_2$ |

# Defining formats

```
// Format
format Name = format

// Format template
format Name parameter_1 … parameter_n = format
```

# File format declarations

```
// The named file shall have the given format.
name :: format
```

# Biksel File Formats — Quick Reference

## Basic formats

```
// Primitive formats:
//
//   Byte     (8-bit unsigned integer)
//   Int32    (32-bit signed integer)
//   Int64    (64-bit signed integer)
//   Float64  (64-bit floating point number)
//
// All primitive formats use little-endian byte order, are
// naturally aligned relative to the beginning of the file, and
// are preceded by zero-valued padding bytes as necessary for
// alignment.
//
// Signed integers are encoded in two's complement
// representation. Floating point numbers are encoded in IEEE 754
// binary representation.

format Bool = "False" | "True"

// Optional value
format Maybe a
   = "Nothing"
   | "Just" >> a

// Sequence of zero or more values
format ArrayOf a
   = ()
   | a >> ArrayOf a

// Self-delimiting list of zero or more values
format ListOf a
   = "Nil"
   | "Cons" >> a >> ListOf a

// Note that 'ListOf a' is usually written '[a]'.
```

# Format identifier

```
format FormatName
   = Byte          // 0: this initial null byte indicates
                   //   immediately to anyone inspecting the file
                   //   that it is not a UTF-8 plain text file

  >> ArrayOf Byte  // format name: sequence of ASCII lowercase
                   //   letters, digits, and hyphens

  >> Byte          // 0: this null byte terminates the format
                   //   name

format FormatId
   = FormatName
  >> Int32       // format major version: non-negative
  >> Int32       // format minor version: non-negative
```

# Sprite file

```
format Color
     // transparent
   = Byte >> Byte >> Byte >> Byte  // 0, 0, 0, 1
     // rgb
   | Byte >> Byte >> Byte >> Byte  // blue, green, red, 0

format Pixmap
   = Int32          // width: non-negative
  >> Int32          // height: non-negative
  >> ArrayOf Color  // pixel values:
                    //   no. elements = width * height

// The order of the pixel values in the above format is as
// follows: start from the bottom-left corner and proceed in
// horizontal scanlines; the pixels within each scanline are
// ordered from left to right; the scanlines themselves are
// ordered from bottom to top.

format Sprite
   = Pixmap
  >> Int32 >> Int32  // (x, y) offset of bottom-left corner of
                     //   pixmap relative to sprite origin

// Recommended file extension: .sprite
format SpriteFile
   = FormatId  // "biksel-sprite" 1.0
  >> Sprite
```

# Biksel Shell Commands — Quick Reference

**`bikbuild [--run]`**

Build program from C source files in working directory. If `--run` is specified, immediately run the program if the build is successful.

**`bikrun`**

Run program from build output in working directory.

**`bikpaint`**

Run image editor.

**`bikpath COMPONENT`**

Print the path of a component of the Biksel system to standard output. COMPONENT must be one of the following:

| | |
|---|---|
| `manual-html` | user manual in HTML format, index.html file |
| `manual-pdf` | user manual in PDF format |
| `examples` | directory containing the example programs |
| `copying` | GNU General Public License in text format |
| `copying.lesser` | GNU Lesser General Public License in text format |
| `source` | directory containing the source code for Biksel |
| `emacs-init-file` | template Emacs init file |

**`bikinfo ITEM`**

Print information about the Biksel system to standard output. ITEM must be one of the following:

| | |
|---|---|
| `version` | version number of the Biksel system |

**`bikpackage ACTION`**

Query or manipulate the Biksel installation. ACTION must be one of the following:

| | |
|---|---|
| `list-files` | print paths of installed files and directories to standard output, one per line |
| `uninstall` | uninstall Biksel; may require superuser privileges depending on how you installed Biksel; will prompt you to confirm before removing any files |

Note: this command is not applicable to some installations of Biksel, e.g. if you installed Biksel via your operating system's package manager.

# Biksel Emacs Lisp Interface — Quick Reference

To load the Biksel Emacs Lisp module, use:

```
(load "biksel")
```

## Commands

**`(biksel-utility-scroll-up)`**

Scroll up a few lines in the selected window.

**`(biksel-utility-scroll-down)`**

Scroll down a few lines in the selected window.

**`(biksel-utility-compile)`**

If the default directory of the current buffer has the extension “`.bik`”, build Biksel program; otherwise, same as “`(compile)`”.

**`(biksel-run)`**

Run Biksel program.

**`(biksel-build-and-run)`**

Build and run Biksel program.

**`(biksel-paint)`**

Open Biksel image editor.

## Variables

**`biksel-utility-scroll-distance`**

Number of lines to scroll with the commands `biksel-utility-scroll-up` and `biksel-utility-scroll-down`. You may change the value of this variable using `setq`. Must be a positive integer.