Robowaifu Design Document

t. The /robowaifu/ Community
Chobitsu, general editor
English translation
version v200428

Abstract

This document describes the specification, design, engineering, implementation, capabilities, and usage of the *Robowaifu Reference Model A Series*. It is intended for designers, technicians, engineers and developers rather than for general lay audience consumption.

This manual is a living research and design document, and is in addition to the general ***Robowaifu User's Guide***. It will be constantly changing and being updated for the forseeable future. Each published revision will have a unique version number (based on date). The **/robowaifu/** community at large (as well as it's neighboring communities) contributed to the design ideas and specifications herein. Without them, this work would not have been possible.

Contents

I. Basic assumptions

This part of the document is intended to be more or less read in order from start to finish. There is nothing complicated here, and information considered fundamental to the rest of the manual is included here.

Here we cover some of the important basics. Where does one even *begin* creating a robowaifu? Why, at the **beginning** of course!

1 Introduction

1.1 Implementer requirements

Every form of robowaifu that is created has at least two human roles involved: **Engineer**s, and **Designer**s. These may be the same man, but usually many men will be involved in teams, with roles often overlapping.

Engineer: The **engineering** will be crafted by the engineer.

TBD.

Designer: The **designs** will be crafted by the designer.

TBD.

1.2 Engineering requirements

When an engineer works to craft a robowaifu, numerous mechanical, electrical, computing, networking and other considerations must be taken into acount.

1. The robowaifu engineering has to be defined. This is basically a job for a professional engineer, but for now Anon will have to step in.
2. The engineering has to be detailed with specifics inside engineering documents so it can be manufactured reliably.
3. *TBD.*

1.3 Design requirements

When a designer works to craft a robowaifu, numerous aesthetic, social and other considerations must be taken into acount.

1. The robowaifu design has to be defined. This is basically a job for a professional designer, but for now Anon will have to step in.
2. The design has to be detailed with specifics inside design documents so it can be manufactured reliably. §18 contains more information about the software used to create these design documents.
3. *TBD.*

→ § 18 layout

1.4 Sociological considerations

A **Robowaifu** has many sociological implications for her **Master**, and the designer and engineer must both take all these points into account when devising the functioning of the robowaifu's varied systems.

1. The social engagement for the master should be pleasing to him.
2. The social engagement for the master should be productive for him.
3. The social engagement for the master must not be harmful to him.[1]
4. *TBD.*

[1] This is not to say the robowaifu must never *admonish* her master when circumstances warrant it. For example, in matters of personal health & safety. To neglect to do so would in fact *be* harmful to her master. Just don't let her turn into some kind of nagging bitch. Therefore, some type of 'off' switch for this aspect is needed.

2 The art of robowaifu engineering & design

2.1 Common rules

There are few common rules because every kind of robowaifu has different requirements and meets a particular **Anon**'s need. Her design should therefore take into consideration *who* will own this specific model of robowaifu, and *why*.

An important criteria for consideration is if her master will live with the robowaifu with her serving him as a mere domestic (a catgirl meido), or if he wants to find a true partner with the robowaifu in 'marital' bliss (a true waifu) or some other variation (for example as just a simple chatbox).

In addition to that, the designer of the robowaifu has to take into consideration and accomodate certain other domestic conventions, like the living habits (sexual and otherwise) of her master. [2]

! → The main purpose of a robowaifu design is to ensure the master finds the satisfaction he wants with her, and to be able to enjoy and fully engage with his pretty robowaifu. However, for the design *documents*; structure, readability, and consistency is far more important than "being artsy". [3]

2.1.1 Technical design documents

The following rules of thumb should be valid for most documents:

Line spacing : The spacing between two lines should be larger than the spacing between two words to guide the eyes of the reader.

Line length : The length of a line – or when using multicolumn layout of a column – should be about 60 characters. When lines get longer they are more difficult to read and it is easier to go to the wrong line after finishing the current one. Increasing the linespacing may help a little. On no account should code block lines exceed 80 characters. Conversely, when lines get too short it is difficult to set them as justified, and you will get lots of hyphenated words.

Page layout : Normal text pages should look the same throughout the document. Figures, tables and special pages like the index need not appear in the same layout but should take as much space as needed.

Margin notes : Margin notes are often more suitable than footnotes because they appear right next to the text they refer to. Special margin notes are the "attention sign" or the "dangerous bend" that guide the user to important parts of the text.

Headings and Footings : Headings and footings should make it easier for the reader to orient himself in the document. If you expect readers to copy single pages from the document they should contain information about the paper as a whole, just in case you need more information or want to cite the whole paper.

If you expect the document to change often (like this one will), each page should contain version information or at least a date.

[2] Compare the lifestyles of different types of anons like "Outdoorsmen" and "Hikkis".

[3] Of course, the robowaifu *herself* should be beautiful, that's an important part of what makes her superior. But you, as a designer, should strive for engineering-grade quality in your documents, not fancifullness. We can save that for marketing.

2.2 Special note for technical-description style documents

Reference Manual Style : This LaTeX design style is suited for our purposes in general, and is used in this robowaifu design document. We recommend its use. [4]

- The text is printed in rather short lines in the right part of the page. This part is used for continuous reading.
- The wide left margin is used for headings and margin notes. Since you now have a wide margin it is easier to use long margin notes to supply additional information and to lead the reader to important parts of the documents. Please note that the margin is always on the left side thus two-sided printing does not look symmetrical. This is done on purpose, because the reader will always start reading at the left side, and with this layout section headers really “stand out”. In a symmetrical layout, half the headers would be buried in the text.
- Figures and tables are either inside the text column, inside the margin or, if necessary, fill the whole page.

→ § **18.6** refman ex.

§ 18.6 describes how to implement such a layout in LaTeX.

[4] The “PostScript Reference Manual” is another document that uses such a design.

II. Physical considerations

Here we'll look at the physical, mechanical side of robowaifu engineering.

This part contains mechanical and wiring diagrams, engineering specs, and other technical descriptions and documentation.

3 Physical safety General

TBD.

3.1 Human safety

TBD.

3.2 Robowaifu safety

TBD.

3.2.1 Avoiding environmental hazards

TBD.

3.2.2 Self-preservation vs. self-sacrifice

TBD.

3.3 Environment safety

TBD.

3.4 Detecting mechanical error-conditions

TBD.

4 Mechanical

TBD.

4.1 Kinematic considerations

TBD.

4.1.1 Structural geometry considerations

TBD.

4.1.2 Mass is everything

TBD.

4.1.3 Thrown mass

TBD.

4.1.4 Inboard mass

TBD.

4.1.5 Bipedal locomotion

TBD.

4.1.6 Eye-hand coordination, &tc.

TBD.

4.2 Armatures & frameworks

TBD.

4.2.1 Head

TBD.

4.2.1.1 Face

TBD.

4.2.1.1.1 Eyes & eyelids

TBD.

4.2.1.1.2 Eyebrows

TBD.

4.2.1.1.3 Mouth

TBD.

4.2.1.2 Neck

TBD.

4.2.2 Torso

TBD.

4.2.3 Arms

TBD.

4.2.3.1 Shoulder joints

TBD.

4.2.3.2 Elbow joints

TBD.

4.2.3.3 Wrist joints

TBD.

4.2.4 Hands

TBD.

4.2.4.1 Palm

TBD.

4.2.4.2 Thumb

TBD.

4.2.4.3 Fingers

TBD.

4.2.5 Legs

TBD.

4.2.5.1 Hip joints

TBD.

4.2.5.2 Knee joints

TBD.

4.2.5.3 Ankle joints

TBD.

4.2.6 Feet

TBD.

4.2.6.1 Toes

TBD.

4.3 Actuators, force-transfer, & control mechanisms

TBD.

4.3.1 Electrical motors

TBD.

4.3.2 Cable-based actuators

TBD.

4.3.2.1 Bowden cables

TBD.

4.3.2.2 Simple (pull) cables

TBD.

4.3.3 Screw (linear) actuators

TBD.

4.3.4 Lever (pushrod) actuators

TBD.

4.3.5 Pneumatic (artificial muscle) actuators

TBD.

4.3.6 Hydraulic actuators

TBD.

4.3.7 Unconventional actuators

TBD.

4.4 Lubrication & joints

TBD.

4.4.1 Fluid-distribution manifolds, pipes, & tubing

TBD.

4.4.2 Circulatory pumps & mechanisms

TBD.

4.4.3 Filtering & cooling

TBD.

4.4.4 Reservoirs

TBD.

4.4.5 Access & replenishment

TBD.

4.4.6 Joints servicing & replacement

TBD.

5 External shell & 'skin'

TBD.

5.1 Hard-surface materials

TBD.

5.2 Soft-surface materials

TBD.

5.3 External shell & cooling-systems symbiosis

TBD.

6 Cooling

TBD.

6.1 Passive cooling

TBD.

6.1.1 Materials

TBD.

6.2 Active cooling

TBD.

6.2.1 Fans

TBD.

6.2.2 Refrigeration

TBD.

6.3 Cooling-fluids & distribution

TBD.

6.3.1 Fluid-distribution manifolds, pipes, & tubing

TBD.

6.3.2 Circulatory pumps & mechanisms

TBD.

6.3.3 Reservoirs

TBD.

6.3.4 Access & replenishment

TBD.

7 Power

TBD.

7.1 Power aquisition & production

TBD.

7.1.1 Household current

TBD.

7.1.2 RF induction

TBD.

7.1.3 Generative braking

TBD.

7.1.4 Solar trickle

TBD.

7.2 Power switching systems

TBD.

7.2.1 Automatic & failover switching

TBD.

7.2.2 Manual switching

TBD.

7.3 Power storage

TBD.

7.3.1 Batteries

TBD.

7.3.1.1 LiPo

TBD.

8 Electrical, controls, & wiring

TBD.

8.1 Electrical

TBD.

8.1.1 Safety

TBD.

8.1.2 Busses

TBD.

8.1.3 Wiring

TBD.

8.1.4 Transformers

TBD.

8.1.5 Inverters

TBD.

8.2 Control

TBD.

8.2.1 Electronics

TBD.

8.2.1.1 Analog

TBD.

8.2.1.2 Digital

TBD.

8.2.2 Relays

TBD.

8.2.2.1 Light-duty

TBD.

8.2.2.2 Heavy-duty

TBD.

8.3 Networking

TBD.

8.3.1 Network gear

TBD.

8.3.2 Network wiring

TBD.

8.3.3 Wireless networking

TBD.

9 Computing resources

TBD.

9.1 Onboard computing resources

TBD.

9.1.1 Single-board computers (SBCs)

TBD.

9.1.1.1 Beaglebone Blue

TBD.

9.1.1.2 RaspberryPi & clones

TBD.

9.1.2 Microcontrollers

TBD.

9.1.2.1 Arduino Nano

TBD.

9.2 External computing resources

TBD.

9.2.1 Home servers & clusters

TBD.

9.2.2 Internet cloud providers

TBD.

10 Sensors

TBD.

10.1 Visual

TBD.

10.1.1 Stereo & multipoint

TBD.

10.1.2 Acuity

TBD.

10.1.3 Infrared & ultraviolet considerations

TBD.

10.2 Audio

TBD.

10.2.1 Stereo & multipoint

TBD.

10.2.2 Distinctiveness & spatial discrimination

TBD.

10.2.3 Frequency response considerations

TBD.

10.3 Tactile

TBD.

10.3.1 Haptic-response systems

TBD.

10.3.2 Sensorimotor & inertial

TBD.

10.3.3 Overall system responses

TBD.

10.4 Other sensors

TBD.

10.4.1 SONAR

TBD.

10.4.2 LIDAR

TBD.

10.4.3 GPS

TBD.

10.4.4 EM & RF, other

TBD.

10.4.5 Particulate & 'olfactory'

TBD.

11 Mechanical construction approaches

TBD.

11.1 Safety

TBD.

11.2 Reading & creating mechanical schematics

TBD.

11.3 Using the right tools

TBD.

11.3.1 Hand tools

TBD.

11.3.2 Power tools

TBD.

11.3.3 Technical instruments

TBD.

11.3.3.1 Multimeters, power, & electronics

TBD.

11.3.3.2 Micrometers & lengths

TBD.

11.3.3.3 Scales & masses

TBD.

11.4 Designing, creating, & using construction rigs

TBD.

11.4.1 Using woods

TBD.

11.4.2 Using plastics, synthetics, & rubber

TBD.

11.4.3 Using metals

TBD.

11.4.4 Using pipes, cables & ropes

TBD.

11.4.5 Presses & other deformation rigs

TBD.

11.5 Manufacturing parts

TBD.

11.5.1 3D printing

TBD.

11.5.2 Choosing stocks

TBD.

11.5.2.1 Plastics, synthetics, & rubber

TBD.

11.5.2.2 Metals

TBD.

11.5.2.3 Fabrics & cords

TBD.

11.5.3 Ensuring compatibility

TBD.

11.6 Fastening parts together

TBD.

11.6.1 Screws, rivets, & mechanical fasteners

TBD.

11.6.2 Glueing & bonding

TBD.

11.6.3 Welding & brazing

TBD.

11.7 Wiring parts together

TBD.

11.7.1 Electrical connections

TBD.

11.7.1.1 Electrical tapes

TBD.

11.7.1.2 Wiring-nuts & other fasteners

TBD.

11.7.2 Electronics connections

TBD.

11.7.3 Soldering

TBD.

11.7.4 Crafting & securing wiring-harnesses

TBD.

11.7.4.1 Safety conduits

TBD.

11.7.4.2 Rigidity considerations

TBD.

11.7.4.3 Range-of-motion considerations

TBD.

11.7.4.4 Servicing considerations

TBD.

11.8 Functional testing

TBD.

11.9 Stress & load-testing

TBD.

11.9.1 Testing individual components

TBD.

11.9.2 Testing subsystems

TBD.

11.9.3 Testing the overall system

TBD.

11.10 Fit & finish

TBD.

III. Software considerations

Here we'll look at the software side of robowaifu engineering.

This part contains code samples, diagrams, engineering specs, and other technical descriptions and documentation.

12 Software safety General

TBD.

12.1 Human safety

TBD.

12.2 Robowaifu safety

TBD.

12.2.1 Avoiding environmental hazards

TBD.

12.2.2 Self-preservation vs. self-sacrifice

TBD.

12.3 Environment safety

TBD.

12.4 Software error-detection & fault-tolerance

TBD.

13 Software construction approaches

TBD.

13.1 The Python programming language

TBD.

13.2 The C++ programming language

TBD.

13.3 The C programming language

TBD.

13.4 IDEs

TBD.

13.5 Software-testing systems

TBD.

13.6 Debugging

TBD.

14 Realtime performance considerations

TBD.

14.1 Realtime subsystems

TBD.

14.2 Non-realtime subsystems

TBD.

15 IPCNet

Inter-Process Communication Network Library.

TBD.

15.1 Introduction

Q&A :

What is the project? IPCNet is a proposed parallel computing library that provides an easy-to-use interface for processes implemented in different languages across many platforms to share their data and provide data services to each other.

Why create this project? Developers require a way to focus on building projects they find interesting and affordable to make, while others can quickly drop these components into their own robowaifu project.

What do processes control? Processes have control over their data and subroutines and define the permissions and rules which they may be accessed, modified and executed. Processes may implement their own whitelists and blacklists, rate limits and usage limits. When processes query or request data it is just that, a request. Other processes do not have to fulfill it.

What is it intended for? IPCNet is intended for parallel processes working together in collaboration, such as the components of a robowaifu plugged into each other. IPCNet is not intended for programs with tightly coupled dependencies. When a process requests data there may be multiple processes available to fulfill the request.

What is the main focus? The main focus is to connect developers' projects together and unite them with a single interface.

What's different? IPC libraries that exist are specific to one or two programming languages and are not easy for developers to quickly include into their projects and interface processes across different languages or even different platforms. Parallel computing and grid computing software is often tooled for special purposes and are not intended for general use either. IPCNet provides a way for all kinds of processes and systems to interact, safely and quickly.

Development Philosophy :

Simple design IPCNet should be simple in its implementation and its interface. The interface should work the same across all platforms and programming languages and parse communicated data into a proper format that the receiving process can readily use. The library and processes that use it should not obfuscate their subroutines beyond what is sensible for security and failsafety.

"Relying on complex tools to manage and build your system is going to hurt the end users. [...] If you try to hide the complexity of the system, you'll end up with a more complex system. Layers of abstraction that serve to hide internals are never a good thing. Instead, the internals should be designed in a way such that they NEED no hiding." — **Aaron Griffin**

Robust and fail-safe IPCNet needs to be robust, fault tolerant and fail-safe. Data coming in through various input methods may contain errors from interference or malice and need to be corrected or discarded. Errors and failures should be handled gracefully and degrade the system in a way that will cause minimal harm to the equipment, other equipment, to the environment and to people. The library and processes implementing the library should consider the following questions:

1. How critical is the component/process?
2. How likely is the component/process to fail?
3. How can the component/process be made fault tolerant?
4. How can the component/process be designed to reduce impact and damage to the rest of the system in the event of failure?

TBD.

15.2 Data requirements

TBD.

Data Transfer : All forms of data should be transmittable over the network with low-latency and high throughput. Some forms of data that need careful consideration:

- Function parameters passed on the call-stack
- General data structures, such as stack and heap C-arrays
- AI tensor data (different libraries need to be able to exchange tensor data)
- Streaming video, raw and encoded
- Streaming audio, raw and encoded
- Other sensor data, such as haptic sensors, SONAR, LIDAR &tc.
- General files

Beyond function parameters, data types will need to be properly annotated with further metadata, describing the encoding of the data so it can be properly decoded and read by other processes.

Data validation : All data transmitted over the network needs to be checked for integrity. It must be free from errors, both ones caused by potential interference sources, and ones devised through malicious intent. It is expected that ongoing pentesting will be a normal, regular part of the robowaifu engineering processes.

Data should be traceable to its source and (optionally) cryptographically tagged historically, indicating which processes/devices modified it, and in what order, so that data pipelines can be quickly inspected and debugged. This approach needs to function under normal runtime conditions to allow for realistic validations.

Modularity : Processes connected together through IPCNet should be easily separated and recombined, with the benefit of flexibility and variety of choice in data and service providers. However, in the spirit of simple design these processes should not hide their complexity behind the interface beyond what is reasonable for security and safety, to ensure the reusability and versatility of these programs.

Fault tolerance :

- No single point of failure – If a system experiences a failure, it must continue to operate without interruption during the repair process.
- Fault isolation to the failing component – When a failure occurs, the system must be able to isolate the failure to the offending component.
- Fault containment to prevent propagation of the failure – Some failure mechanisms can cause a system to fail by propagating the failure to the rest of the system. Mechanisms that can isolate a rogue component or failing component when it is unable to contain a fault are required to protect the system.
- Availability of reversion modes – the abandonment of one or more recent changes in favor of a return to a previous version. If a component update breaks the system, it should revert back to the last working version.

15.3 Security

If a malicious entity successfully alters the data of an IPC socket, it/they are effectively tampering with a low-level mechanism part of the core operating system. If an attacker is able to do this–or even if they just manage to intercept the internal data–then that device has already been compromised.

Any processes with direct access to the Internet could potentially be exploited to create malicious requests within the IPC network or even become fully compromised by arbitrary code execution. Direct connection of the robowaifu with the Internet–or bridged connections out the Internet–is therefore highly discouraged. Use air-gapped systems at the least. Strongly validated data sources, &tc. are instead encouraged.

Additionally, any process could connect to the IPCNet and send malicious requests not created through the IPCNet library. Potentially rogue processes need to be identified, mitigated and isolated. Inputs from requests need to be sanitized and bad requests refused.

Permissions : Process routines registered on the network should have explicitly defined read, write and execution permissions for processes and process groups.

TBD.

Process Groups : Processes can register processes into groups and give them their own permissions.

TBD.

Denial of Service mitigation : *TBD.*

- Identify normal conditions for network traffic
- Filter malicious traffic (connection tracking, component reputation lists, black/whitelisting, or rate limiting)

TBD.

15.4 Compromised systems

If a robowaifu subsystem is compromised by an attacker there is no reasonable way to verify the trustworthiness of processes thereafter. Data may be modified or processes abruptly shutdown by an attacker with intent to cause damage or harm.

This effectively implies the entire robowaifu system is compromised as well. Automatic checks (and backups) should be in place to detect compromised states and immediately shutdown gracefully & safely. Diagnostic information similar to the infamous BSOD must be made available to the robowaifu's Master to assist with the technical remediation efforts.

TBD.

15.5 Feature set

General features : *TBD.*

- P2P networking for processes – connected processes form a mesh network and can communicate with each other even if they're not directly connected.
- Announce data – processes can announce to the network their data and data services.
- Find data and data services – processes can search the network for data and other processes providing data services.
- Query data – processes can query data that other processes already have in memory or on disk and have made available.
- Subscribe to data and data services – processes can subscribe to data and receive new updates whenever it is changed, or subscribe to services that provide a stream of data, continuously or at regular or irregular intervals.
- Offer service – processes can provide data services to other processes to execute and produce new data on request. These services may be public or private.
- Request service – processes can request data providers to execute subroutines that produce new data.

15.6 Implementation

TBD.

Booting :

Event-driven interface :

Functions & returns :

Arguments & parameters :

Metadata :

IPC sockets :

Network sockets :

Security :

Status Codes :

- Success
  - 100 OK
  - 101 Created
  - 102 Accepted
  - 103 No Data
  - 104 Partial Data
- Client errors
  - 200 Bad Request
  - 201 Unauthorized
  - 202 Forbidden
  - 203 Not Found
  - 204 Method Not Allowed
  - 205 Too Many Requests
  - 206 Upgrade Required
- Service errors
  - 300 Internal Process Error
  - 301 Not Implemented
  - 302 Bad Gateway
  - 303 Service Unavailable
  - 304 Gateway Timeout
  - 305 IPCNet Version Not Supported
  - 306 Insufficient Storage
  - 307 Insufficient Memory
  - 308 Loop Detected
  - 309 Network Authentication Required

15.7 Discussion, further Q&A

TBD.

1. How can we gracefully shutdown the system? A robowaifu may need to move to a safe location and shutdown in a way that does not cause damage to the components, other equipment near it, the environment or nearby people.

2. How can we gracefully shutdown a compromised system? An attacker who has compromised a device in the network may modify data or shut it down with intent to cause damage or harm.
3. How critical is the component/process?
4. How likely is the component/process to fail?
5. How can the component/process be made fault tolerant?
6. How can the component/process be designed to reduce impact and damage to the rest of the system in the event of failure?
7. How can we gracefully shutdown the system?
8. How can we gracefully shutdown a *compromised* system?

16 Control & communications UI

TBD.

16.1 Onboard C&C UI

TBD.

16.2 External C&C UI

TBD.

17 AI & Machine Learning

TBD.

17.1 Foundations, briefly

TBD.

17.2 Current AI/ML frameworks

TBD.

17.2.1 TensorFlow

TBD.

17.2.2 PyTorch

TBD.

17.2.3 Keras

TBD.

17.3 Compute-resource considerations

TBD.

17.4 3rd-party APIs & cloud-based systems

-Note: This subsection will be deleted entirely in the future. -Chobitsu

While we don't recommend this approach in general, and certainly not in the long-term, the simple fact is that for *today*, this approach is the most expedient way for a robowaifu technician to include relatively sophisticated AI capabilities into his robowaifu. While the learning curve required for these tools varies, in each case it is substantially less than that required for implementations 'rolled by hand'.

The mandatory cautionary notes:

- **Invasive** – your privacy is compromised. You have no control (or even knowledge of) what is done with your data (for example, the questions you may ask your robowaifu). Whatever it is, you can be assured *it won't serve your best interests*, but rather the provider's and those they answer to (for example; governments, investors, and secondary 3rd-party commercial partners). Remember the old adage: "When something is free, *you* are the product".
- **Fragile** – you can be deplatformed (or worse) at a moment's notice for wrongthink, etc. Remember Tay.ai, as well as the many and overt abuses by companies such as YouTube, Facebook & Twitter?
- **Survelliance state** – your data can be (and likely *is* being) submitted to the authorities–by law–when you use these services.

With these highly-negative consequences in mind, we strongly urge you to only consider these vendor services simply as a stop-gap measure if you must use them, and to seek more permanant solutions that are both open and far more private, secure, and stable for you and your robowaifu in the future. Rest assured you are not alone in this and that we are working towards solutions for these basic needs together as a unified community.

If all this isn't enough to dissuade you, then be warned a last time: remember, **it is in the commercial & other interests of these vendors to have you become addicted to their services**. Over time, it will become increasingly easier for you to compromise on your principles, and to simply ignore the fundamental evils and disservice happening to both you and your robowaifu 'behind the scenes' by using these facilities.

Fair warning then, Anon.

17.4.1 Wolfram|Alpha

TBD.

https://products.wolframalpha.com/api/

17.4.2 Microsoft CNTK

TBD.

https://docs.microsoft.com/en-us/cognitive-toolkit/

17.4.3 Amazon AI Services

TBD.

https://aws.amazon.com/machine-learning/ai-services/

17.4.4 Google AI

TBD.

https://aihub.cloud.google.com/

17.5 Our general robowaifu ML approach

TBD.

17.5.1 Reinforcement Learning (RL)

TBD.

17.5.2 Deep Neural-Networks (DNNs)

TBD.

IV. Design considerations

Here we'll look into some of the aesthetic and other aspects of robowaifu designs.

This part contains design diagrams, design specs, and other design descriptions and documentation.

V. Reference, appendices, etc

This part is a grab-bag of general reference information.

18 How to change a LaTeX layout?

18.1 Advantages and disadvantages of LaTeX

Advantages: The big advantage of LaTeX is, that it implements a "generic" or "logical" design. This means that the author has to specify the *meaning* of special parts of the text like: headings, citations, lists, literature references, and so on. These logical definitions will be processed by the system and printed in the "correct" way. The meaning of *correct* is defined in the document class and additional packages.

The opposite of this is the "visual" design that most text processors use. Here the author has to know the correct way to set certain parts of the text and take care of the correct printing.

The logical design makes it easier for the author to write consistent documents (i. e., same font and fontsize for section headings of the same level, same layout for lists and enumerations, . . .).

LaTeX is powerful and flexible: you can define arbitrary document design by changing the definitions in the class files or overwrite them with packages. This is easier than you probably think.

Disadvantages: The main disadvantage of LaTeX was that the author originally had only limited means to change the layout and that he had only four classes to choose from. This changed a great deal with the improved Koma-Script classes for LaTeX 2_ε and today it's easy to use with a small learning curve.

18.2 Input files and class files

According to the principle of separation of content and design, there are two kind of files:

- The content and the logical structure of a document are defined in the LaTeX input file.
- The design (layout) is defined in the class files and packages.

Which class and packages files a document will use is defined at the beginning of the input file. The `\documentclass` command selects the class and the `\usepackage` command specifies additional packages.

To generate a document you need at least two files, the input file and a class file.

- Similar documents (that appear in a series) have the same layout because the layout is defined in a file of its own and not part of the document (this is similar in concept to the way CSS works).
- You can output the same content without much rework using different layouts, e. g., as an article for a magazine and as a chapter for a dissertation.

18.3 Class files and packages

LaTeX supports a hierarchy of layout definitions.

- The first file processed is the class file that is specified inside the curly braces of the `\documentclass` command. This defines the kind of document you want to write.

- The optional argument of the `\documentclass` command inside the square brackets defines class options which select variants of the basic layout, such as different font sizes.
- The last step is reading the packages specified by the `\usepackage` command. This command again takes options to select the layout.
- You can change layout parameters in the input file, but this is discouraged because it violates the principle of separation of content and design.

There are some important differences between class and package files and "normal" input files:

- Class and package files should only contain definitions. They must not output text.
- The "at"-sign `@` is treated as a letter and therefore may appear in command names. Most internal commands of LaTeX contain an `@` to prevent the author from using them accidentally.
- The extension of the file is `.cls`, `.clo` or `.sty` instead of `.tex`.

18.4 Changing the layout, step by step

! :-) First off Anon, it's much simpler to begin with an already-existing document that is close to your needs (e. g., just making a copy of and modifying this robowaifu design document would be a good place to start, and then you can basically just skip the rest of this section).

-It's how I did it, using the refman LaTeX package. -Chobitsu

It also is usually easier to change existing class files instead of writing a new one from scratch. In many cases it is even sufficient so replace some definitions and put them into a package instead of creating a new class.

Please note that you are *not* allowed to change the standard classes distributed with LaTeX. You *have* to change the name when you want to make changes. That is another reason to put small changes in packages.

18.4.1 Differences between desired and available layout

The first step is to determine the difference between the layout you have and the layout you want.

18.4.2 Finding the original definition

The next step is to find out where the original layout is defined. It is best to search the files in the following order:

1. the LaTeX manual by Leslie Lamport,
2. the LaTeX documentations files `*.dtx` for the classes or packages
3. the LaTeX documentations files `*.dtx` for the kernel,
4. the TeXbook by Donald E. Knuth.

The files are usually documented quite well so you should be able to change things even if you don't understand everything.

18.4.3 Writing a new package file

The third step is to create a new package. You choose an appropriate name for the package (like `mysty`) and create a filename by adding the extension `.sty`.

This file will only contain the definitions you want to change or the new commands you want to define.

If you want to change definitions or certain parameters, the best way is to copy them from the original file and modify them according to your liking.

Defining new commands is easier when you find similar commands in the original files which you can change.

It is always a good idea to include the reason you wrote the package, the changes it makes and the new commands it defines in the file. You should include the date of the last change and the LaTeX version it works with, just in case some internal LaTeX commands you use will change.

When writing larger packages, it is an even better idea to use the `docstrip` program which is used to document the LaTeX 2ε files. Thus you have your code and documentation in one file and it's easier to keep them from going out of sync.

18.4.4 Using the new package

To use the new package, you call it with the `\usepackage` command. This command executes the code of your package and changes the layout as desired.

Example:

```
\documentclass[11pt,twoside,a4paper]{article}
\usepackage{mysty} %<- This calls the package "mysty"
```

You shouldn't need to change anything else in your input file, unless you defined new commands or environments that are not available in standard LaTeX.

! → When you copy your input file to a different computer you have to include your new packages as well. Otherwise the document can't be processed.

18.5 A simple example (Equation numbers)

Let's assume that you want to write an article where the equations are numbered separately in every section. In the LaTeX manual you find a notice that the report class does something similar for every chapter.

Looking into the file `report.cls` you will find the following commands that deal with equation numbers:

```
\@addtoreset{equation}{chapter}
\def\theequation{\thechapter .\arabic{equation}}
% or in LaTeX2e since 1995/06/01:
\renewcommand\theequation{
             \thechapter.\@arabic\c@equation
                             }
```

You don't necessarily need to understand these two commands in detail.

Now you create an new file with the name `eqpersec.sty`[5] and copy the commands above into that file. After that you replace every occurrence of `chapter` with `section` and add some comments.

```
% This is equation_per_section.sty
% Short name: eqpersec.sty
% Original file by Hubert Partl 1988
% Modified by Axel Kielhorn 1996/01/01
% to support LaTeX 1995/06/01 and later
%
% reset the equation counter at the start
% of a new section
%
\@addtoreset{equation}{section}
% Equationnumber = sectionnummer.equationnummer
% Use only one of the below
% depending on you LaTeX version
%
%\def\theequation{\thesection .\arabic{equation}}
% or in more recent versions of LaTeX
\renewcommand\theequation{
             \thesection.\@arabic\c@equation
                              }
```

Whenever you use a `\usepacakge{eqpersec}` command as in

```
\documentclass[11pt]{article}
\usepackage{eqpersec}
```

you will get equations numbered according to your conventions.

18.6 A more complex example (Reference Manual)

We want to create a layout similar to the one used in the *PostScript Reference Manual*, with a wide left margin for headings and margin notes and a small margin at the right and bottom.

18.6.1 Page layout

To define the new layout we use the commands described in the LaTeX manual. For full details see the file `refman.dtx`.

Horizontal: First we define two new names for length that we will use often:

`\fullwidth` is the width of the whole page minus a margin of 1 inch on every side.

$$\texttt{fullwidth} = \texttt{paperwidth} - 2\text{ inch}$$

From this the width of the text is calculated.

$$\texttt{textwidth} = \texttt{fullwidth} \times \texttt{textfraction}$$

`\leftmarginwidth` is the width of the left margin that will be used for headings and margin notes.

$$\texttt{leftmarginwidth} = \texttt{fullwidth} - \texttt{textwidth}$$

[5] Depending on the computer you are using the name may be different like `EQPERSEC_STY` on a CYBER running NOS/VE. But note that you must not use spaces in the filename.

This is a little more difficult in reality because the lengths have to be rounded to full points and a possible two column layout – as used in the index – must be taken into consideration.

Vertical: The vertical layout is a little more difficult because you have to deal with the page header and footer.

$$\texttt{textheight} = \texttt{paperheight} - 2.5 \text{ inch}$$

The result of this calculation is rounded to full lines. Depending on the page style – headings or footings – it is shifted up or down by one line.

18.6.2 Section headings

The headings have to be modified to make them extend into the left margin.

In file `classes.dtx` we find the `\@startsection` command that defines the layout of the headings. Only parameters 4 to 6 are relevant for us: parameter 4 is the space above and parameter 5 the space below the section. The 6th parameter does the actual formatting.

This is the original definition:

```
\newcommand\section{\@startsection
             {section}{1}{\z@}%
             {-3.5ex plus -1ex minus -.2ex}%
             {2.3ex plus .2ex}%
             {\normalfont\Large\bfseries}}
```

The commands for sub- and subsubsections are similar. Note that the measures are all in `ex`, thus depending on the font size used.

We define a new command `\secshape` to format the headings. This command uses the whole width of the page for the heading. To discourage hyphenation of the heading we give it a high penalty. This still allows hyphenation when absolutely necessary.

```
\newcommand\secshape{%
               \leftskip=-\leftmarginwidth%
               \rightskip=\@flushglue%
               \hyphenpenalty=2000}
```

This command is inserted into the 6th parameter of `\@startsection`.

Since the headings now extend into the left margin, we can use a smaller font and reduce the space between the text and the heading. The new definition looks like the following:

```
\newcommand\section{\@startsection
             {section}{1}{\z@}%
             {-2ex plus -1ex minus -.2ex}%
             {0.5ex plus .2ex}%
             {\secshape\normalfont\large\bfseries}}
```

18.6.3 Setting the margin notes

The margin notes should always appear on the left side of the text. The normal layout puts them into the outer margin in twoside layout.

The file `latex.dtx` contains the definition of the `\@addmarginpar` command which is responsible for the margin notes. We don't have to understand the whole definition; the important part is the internal variable `\@tempcnta` that is either `\@ne` (1) when the note should appear on the right side of the text or `\m@ne` (−1) when it should appear on the left side.

This is done by the following lines:

```
\@tempcnta\@ne
\if@twocolumn
    \if@firstcolumn \@tempcnta\m@ne \fi
\else
  \if@mparswitch
     \ifodd\c@page \else\@tempcnta\m@ne \fi
  \fi
  \if@reversemargin \@tempcnta -\@tempcnta \fi
\fi
```

which we simply replace by:

```
\@tempcnta\m@ne
```

The remaining lines that handle the setting of the margin note depending on the parameter `\@tempcnta` are left unchanged.

18.6.4 Extensions

The definitions described above are sufficient for simple applications but in practical use one may want some additional commands.

→ Appendix B

You will find the description for the whole new class in the appendix B.

Appendix

A The page structure in LaTeX

→ Fig. 1

This appendix describes how the actual page is build from its components and how they are influenced by TeX's parameters. (Figure 1 has been created by Nelson Beebe at the University of Utah.)

text area : The normal text area ("Body") contains the running text including footnotes, tables and figures. The headings, footer and margin notes do *not* belong to the text area.

The text area has the width `\textwidth` and the height `\text-height`.

In a two column layout the text area is split into two columns, with the width `\columnwidth` each and a space of `\columnsep` between them. Thus the `\columnwidth` is a little bit smaller than half the `\textwidth`.

`\textwidth` and `\columnwidth` should be a multiple of the width of one character in the `tt` font.

`\textheight` should be multiple of the line height `\baselineskip`, increased by the constant value of `\topskip`.

Indentations inside the text area are defined with `\leftskip` and `\rightskip`. These parameters should not be changed explicitly by the user but rather implicitly through environments.

left margin : The left margin is either `\odd-` or `\evensidemargin` plus 1 inch. Both parameters have the same value, unless the `twoside` option is given.

top margin : The top margin is the sum of `\topmargin`, `\headheight` and `\headsep` plus 1 inch.

right margin : The right margin is the paper width minus the left margin and the text area.

bottom margin : The bottom margin is the paper height minus the top margin and the text area.

heading : The heading is inside the top margin with a space of `\headsep` between the lower border of the header and the upper border of the text area. Above the header is a free space of `\topmargin` increased by 1 inch.

footing : The footer is inside the bottom margin with a space of `\footskip` between the lower border of the text area and the lower border of the footer.

margin notes : Margin notes are inside the left or right margin. They have a width of `\marginparwidth` and a space of `\marginparsep` between the margin note and the text area. The vertical space between two margin notes is `\marginparpush`.

The `paperheight` consists of the following elements (from top to bottom):

1 inch
`\topmargin`
`\headheight`
`\headsep`
`\textheight`
`\footskip`
remaining page.

On pages with margin notes in the right margin the `paperwidth` consists of the following elements:

1 inch
`\oddsidemargin` or `\evensidemargin`
`\textwidth`
`\marginparsep`
`\marginparwidth`
remaining page

With the option `twoside` the left pages change to

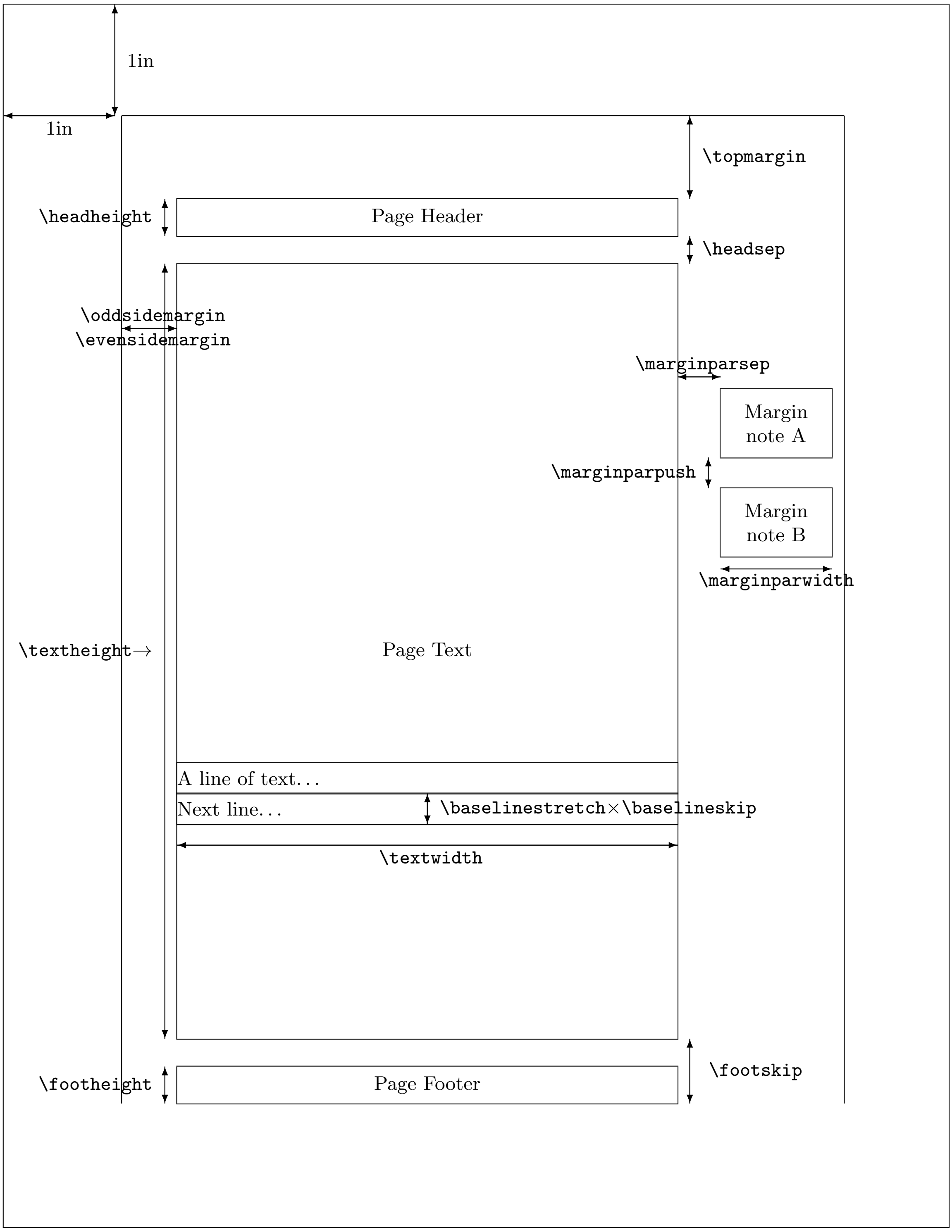
1 inch
`\evensidemargin`
`\textwidth`
remaining page

Comments: The parameters `\topmargin`, `\oddsidemargin`, and `\evensidemargin` may be negative. In this case, the margin will be smaller than 1 inch. The same is true for `\leftskip` and `\rightskip` which leads to text that is wider than the text area.

Extensive treatment and figures to this subject may be found in the TUGBOAT Vol.9, No.1 (April 1988).

The parameter `\footheight` is no longer defined in LaTeX 2_ε since no-one used it.

Figure 1: Page layout



B Description of the refman-class family

The `refman.sty` was defined at the EDV-Zentrum (computing center) of the TU[6] Wien. This layout is suitable for reference manuals, technical descriptions and similar applications. It is based on the ideas shown in previous sections: The layout has a wide left margin for headings and margin notes and smaller margins on the right side, the top and the bottom.

In 1994 this layout was re-implemented as a class for the new LaTeX 2_ε. This made it possible to include some minor improvements, such as the support of different paper sizes. The `refman.sty` was split into two classes `refrep`, similar to `report` and `refart`, similar to `article`. These classes differ in the layout of the header and footer. The `refart` does not support the `\chapter` command.

The current version of both classes is described in this document. It serves as an example for the layout.

B.1 Invocation

The LaTeX local guide (if available) shows if this class is available at your TeX installation or where to install it. To use the `refart` class, simply call it with the `\documentclass` command:

```
\documentclass[11pt,a4paper]{refart}
\usepackage{german} % other packages you may want
```

B.2 Options

The `refart` class replaces `article` and `refrep` replaces `report`. They support all options of these classes except for the `twocolumn` option.

It supports the additional option `square` which makes the `\textheight` equal to the `\textwidth`.

Neither `refart` nor `refrep` support two column layout, thus the commands `\twocolumn` and `\onecolumn` must not be used.

The index will be set in two column format and you can't change it with the means of this class.

B.3 Layout changes

B.3.1 Page design

Horizontal: In this design the usable area for text (`\fullwidth`) is calculated as the paper width minus 2 `\papermarginwidth`. The default value for `\papermarginwidth` is 1 Inch.

The option `smallborder` reduces `\papermarginwidth` to 0.25 Inch. This is more suitable for documents viewed on screen, especially when combined with the `a5paper` and `landscape` options.

Only a fraction of this width is used for the running text (`\textwidth`), the remaining part forms a wide left margin (`\leftmarginwidth`) which is used for headings and margin notes. The `\textwidth` is 70 % of the `\fullwidth` by default, but this can be changed with

[6] Technical University

the `\settextfraction` command which accepts arguments between 0 and 1.

Vertical: The text height is calculated as the paper height minus 2 `\papermarginwidth`. The `topmargin` is modified by some pagestyles. (see B.4.3).

The pages are always set with a ragged bottom.

B.3.2 Section headings

The headings for `\section`, `\subsection`, and `\subsubsection` extend into the left margin, thus using the full width of the page. They are not justified and hyphenation is discouraged. A small space is kept free above and below the heading. Headings for `\section` and `\subsection` are set in a bold font.

The `refrep` class defines a different layout for the `\chapter` command: It always starts a new page and prints the chapter headings in a large bold font with a thick line above and below. This heading uses the full width of the page.

A similar heading is created by the `\part` commands which is available in both classes. It uses a roman part number instead of the arabic section number.

The `\maketitle` commands sets the title of the document in the same layout when no special title page is requested. (This is the default for `refart`. To suppress the title page in a `refrep` document, you can use the `notitlepage` option.) The name of the author and the date is printed in italic flush right below the document title.

B.3.3 Paragraphs

Paragraphs are separated by a vertical space (`\parskip`) of half a line (`0.5\baselineskip`) plus a stretchable length of 2 pt. Paragraphs are not indented.

The vertical spacing inside, above and below a list environment is the same as in the running text.

B.4 Footnotes

The footnote layout consists of a small margin (1em) which contains the footnote symbol. A small space is set between the symbol and the footnote text. The paragraphs of the footnote are not indented. There is currently no space between two footnotes, I'm not sure it this will stay this way. The footnote symbol is set as a superscript. This may change in later versions. I'm relying on user feedback to finally solve this.

B.4.1 Description environment

The `description` environment will use the whole left margin for the description label.

→ Section 18

You will find examples in the section 18.

B.4.2 Positioning of margin notes

Margin notes (`\marginpar`) are always put into the left margin. They use the whole width of the margin.

The minimum space between two margin notes is set to 0 to prevent them from being shifted around when many margin notes are used.

B.4.3 Headers and Footers

The page style `plain` puts the page number into the footer in the right corner. When the option `twoside` is active, the page number of left pages is put into the left corner.

The pagestyles `headings` and `myheadings` create a header which spans the whole width of the page. The headings contain the running head (`\section` and `\subsection` in `refart` and `\chapter` and `\section` in `refrep`) when `headings` is used or a fixed text that can defined with the `\markboth` command when `myheadings` is used. The heading will be set in a slanted font and separated from the body by a thin line.

In addition to the standard classes, `refman` supports a style for footers, which is used in this documentation. The information is exactly the same as in the headings but now printed in the footer with a thin line above.

To use a user-defined string you can say:

```
\pagestyle{myfootings} % or myheadings
\markboth{left title}{right title}
```

The `heading` and `myheading` commands increase the top margin by one line while the `footings` and `myfootings` commands decrease the top margin by one line. The page styles `empty` and `plain` leave the top margin unchanged. You should not combine headings and footings in one document.

User feedback has shown that it is not a good idea to combine `plain` and `(my)heading` either. Therefore I changed the layout of the `\chapter` page to `empty`. Maybe it is necessary to define a `hplain` and `fplain` pagestyle or to define some magic to use the correct definition of `plain`. Feedback is welcome.

B.5 Additional commands

B.5.1 Marginlabel

The command `\marginlabel{xxx}` prints the text `xxx` right justified into the left margin. Please note that a `\marginpar` will print it left justified.

Example: The word “Example” in the left margin is printed with the command `\marginlabel{Example:}`

B.5.2 Attention

! $\rightarrow$ The command `\attention` puts an exclamation mark with an arrow pointing to the text into the left margin. This is an example for `\attention`.

`:-)` Since version 2.0c you can change the symbol used for the `\attention` command using a `\renewcommand{\attentionsymbol}{\texttt\{:-)\}}` command. To get the default back use `\renewcommand{\attentionsymbol}{\large \bfseries ! $\rightarrow$}`

`:-(` Since version 2.0c `\attention` takes an optional argument to define the symbol used in the margin. Thus you can change the symbol once, without having to restore it later. Do not abuse this feature, it is primarily meant as an support for the `manfnt` package which enables you to use the "dangerous bend" and "double dangerous bend" signs.

The `manfnt` package is no longer enclosed with Refman, it has grown and is now a package of its own.

B.5.3 Seealso

The command `\seealso{n}` prints an arrow and its argument into the left margin. You will find examples for this in the left margin and in chapter 1.

→ Chapter 1

B.5.4 Maxipage environment

The `maxipage` environment is a special kind of `minipage` which extends over the full width of the page. It can be used for long formulas or `tabular` environments. You may use `maxipage` environments inside floats. You cannot use margin notes inside a `maxipage` and no page break will occur while in a `maxipage`. A `maxipage` is always a paragraph of its own with a thick line above and below. You can disable these lines with the `\maxipagerulefalse` command. They are on by default.

The following paragraph is an example for a `maxipage`:

This very long line is an example for a `maxipage`. It extends over the full width of the page, including the left margin.

This is normal text after the `maxipage`.

B.5.5 Fullpage environment

The `fullpage` environment consists of one or more pages where the text extends over the full width of the page. You cannot use margin notes inside a `fullpage` environment. A `fullpage` will always start and end on a page of its own. It may be used for large tables, program listings or anything that does not fit into the normal layout.

→ Page 49

Page 49 is an example for a `fullpage`.

B.5.6 Noparskip

The `\noparskip` removes the vertical space between two paragraphs. It is similar to the `\noindent` command that removes the indent of the first line of a paragraph.

B.5.7 Setleftmarginwidth

The `\setleftmarginwidth` command is no longer supported. You can achieve similar results by using the `\settextfraction` command.

B.5.8 Descriptioncolon

By default a colon is printed after the description label. The command `\descriptioncolonfalse` disables the colon, the `\descriptioncolontrue` re-enables it.

B.5.9 Descriptionleft

The `\descriptionlefttrue` command sets the description label left justified into the margin. The default is right justified which will be achieved with `\descriptionleftfalse`

B.5.10 Maxipagerule

You can disable the rules before and after a `maxipage` with the `\maxipagerulefalse` command and re-enable them with the `\maxipageruletrue` command. The default is on. You should not mix `maxipage`s with and without rules in one document.

B.5.11 Condbreak

The command `\condbreak{2cm}` ensures, that the next 2 cm are either completely on this page or completely on the next. No page break will appear in the next 2 cm.

This is really a hack to achieve what the `\samepage` command often fails to do.

B.5.12 Example

The `example` environment acts like a `verse` environment but uses a `tt` font.

B.5.13 Pageperchapter

The command `\pageperchapter` creates page number that start with 1 for every new chapter. This may be useful for larger manuals. Since it works with chapters it is only available in the `refrep` class.

B.5.14 Smallborder

The normal border around the page is 1 Inch. That is fine for a printed document, but wastes a lot of space when a document is meant for reading on screen. The option `smallborder` reduces the margin to 0.25 Inch.

You can redefine the border with `\setlength\papermarginwidth {0.25in}`. Call `\setpagefraction{0.7}` afterwards to recalculate the page layout.

B.5.15 Dvips

The option `dvips` tells DVIPS about the current page size.

B.5.16 Pdftex

The option `pdftex` tells PDFTEXabout the current page size.

B.5.17 Pagesize

`pagesize` chooses the correct `\special`-command to tell the DVI-driver about the paper size. It works with DVIPS and DVIPDFMX for DVI output and PDFTEX for PDF output.

B.5.18 Ifpdfoutput

You can use `\ifpdfoutput{pdftext}{dvitext}` to write different text depending on the output format. This command was necessary to implement the `pagesize` option and is available for the user as well.

The last four commands have been taken from KOMA-Script, thanks Markus.

C Robowaifu programming language tutorials

In the following three tutorial appendices (one for each programming language highlighted in this document; Python, C++, & C), we'll create sample applications as an aid to learning these languages for newcomers.

- First we'll devise an application that 'scrapes' imageboards, then writes the JSON, HTML, & media files from each thread into thread-specific folders on the user's local drive. You can think of this utility as kind of like a basic imageboard archiver once you've finished creating it.

UPDATE: I just found out that julay.world is shutting down. As I was asked to help out with MultiScraper, I may just refocus these tutorials around that effort instead. -t. Chobitsu

These exercises are intended to provide enough of an introduction for each programming language that after completing a language's section, an anon will have the basic understanding of it sufficiently well to begin writing his own software in whichever one he chooses.

C.1 Getting everything ready for software development

Be advised: this process can take days, or even weeks to get right for the beginner. Just be patient with yourself, and you'll get there.

The system & compiler : You'll need both an operating system and a compiler system to write C++ software for your robowaifu. While there are a lot of choices out there, I'll be giving specific tutorial examples here using my own machine, namely **Manjaro Linux** (an *Arch Linux* derivative, and one good for beginners) and the GCC compiler, **g++** (by far the most commonly used C++ compiler on Linux platforms).

Of course feel free to use other combinations that work for you. But for the complete beginner, I'd recommend just following along with the same setup I'm using. It should make a lot of things easier for you once you get going with it, even if there's a bit of a learning curve at first when you're just getting started.

It's beyond the scope of this section to describe the operating system installation process for your machine. I'd recommend you just use the official guidelines for doing so. Installing Manjaro separately on a dedicated machine (if possible) will make your life simpler by not having to deal with mixed platforms. For example, when dual-booting or running under a virtual-machine environment.

BTW, my machine is also using a lightweight desktop environment for Linux called XFCE. If you'd be more comfortable if your environment matches the screen-caps that are here, then I'd recommend you also install the same DE for Manjaro as my system:

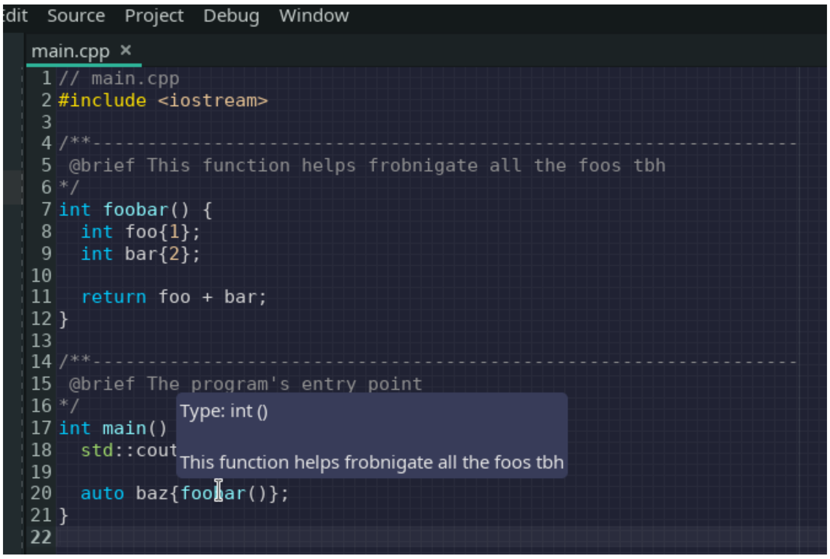
Manjaro XFCE

Manjaro first steps

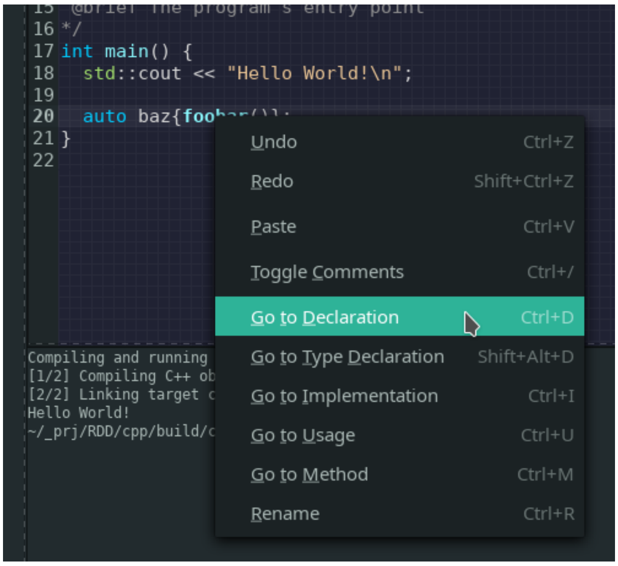
The editor : You need a good way to edit text files. While something as simple as the windows notepad can *technically* suffice, it's far better to use some type of editor that intentionally offers advantages for the software developer, and specifically in this case for the C++/C developer. And as the systems you're working on grow more complex over time, the more you'll need the assistance of a good editing environment to help you with that increasingly-arduous task.

There is a huge array of choices in this area, so rather than waste time attempting some type of comparison here, I'll just give you the examples from my own system. This is how I'm doing it ATM, YMMV. The bottom line: simply find some good, functional way to edit large collections of complicated text files that you're comfortable with, and then gitgud at it Anon.

The most important initial considerations for me is that the editor system be freely available, works on Linux, and preferably be open-source. Emacs and Vim are a couple of good examples that come to mind, for instance. Further, given my desire for systems that really hand-hold the developer, I actually have a preference for integrated development environments (IDEs). Among other features, a good IDE will include some type of 'intellisense' built right in that can help you quickly see a function's type or locate a type's declaration. Here are a couple of example caps of what I mean by this:



and:



The important points to notice in these examples are that in both cases I placed the mouse cursor over the word ***foobar*** on line 20: `auto baz{foobar()};` . In the former case I simply hovered over it (note the b&w 'I-beam'), and in the latter case I right-clicked on it, then moved the cursor down to highlight a context-menu selection.

Hovering over the function name gave me a pop-up rectangle (here in a kind of blue-gray) that tells me both the function's type: `int ()` , as well as any Javadoc brief for it, if present. In this example there is one: *This function helps frobnigate all the foos tbh*

Right-clicking brings up a context menu for an object, and these were the choices available for a `foobar()` for this editor in this code. Left-clicking on that selection would immediately jump to that portion of the codebase, in this case to line 7 of *main.cpp*.

You might ask "Fine, but why do we need this? I can easily just look up at the function itself." Fair point, but ask yourself *what will it be like when you have 200 code-files spread out among numerous, tiered-directory structures?* Being able to easily see an item's (distant) details right there on the spot where you're actually doing the coding ATM can be a big help both for quickly writing good code, and in reducing the likelihood of creating errors in the first place.

TBD.

D Python tutorial

Here's the canonical *Hello World* in Python:

```python
# hello.py               # good to put the filename at top
                         #
print("Hello World!")    # print string to console
```

TBD.

E C++ tutorial

Here's the canonical *Hello World* in C++:

```
// main.cpp                                  // good to put the filename at top
#include <iostream>                          // include the iostream library
                                             //
int main() {                                 // main() function, has int return
  std::cout << "Hello World!\n";             // cout & '<<' operator for string
}                                            // function closes, objs destroyed
```

foobar foobar

```
#include <iostream>
```

barfoo barfoo

```
/**-------------------------------------------------------------------
 @brief This function helps frobnigate all the foos tbh
*/
int foobar() {
  int foo{1};
  int bar{2};

  return foo + bar;
}
```

```
/**-------------------------------------------------------------------
 @brief The program's entry point
*/
int main() {
  std::cout << "Hello World!\n";

  auto baz{foobar()};
}
```

TBD.

F C tutorial

Here's the canonical *Hello World* in C:

```c
/* main.c */                    /* good to put the filename at top */
#include <stdio.h>              /* include the standard IO library */
                                /**/
int main() {                    /* main() function, has int return */
  printf("Hello World!\n");     /* print formatted string to console */
  return 0;                     /* explicit return of success code */
}                               /* function closes */
```

TBD.

G Robowaifu electronics tutorials

H tutorial A

I tutorial B

J tutorial C

K Robowaifu mechanical & construction tutorials

L tutorial A

M tutorial B

N tutorial C

Index