

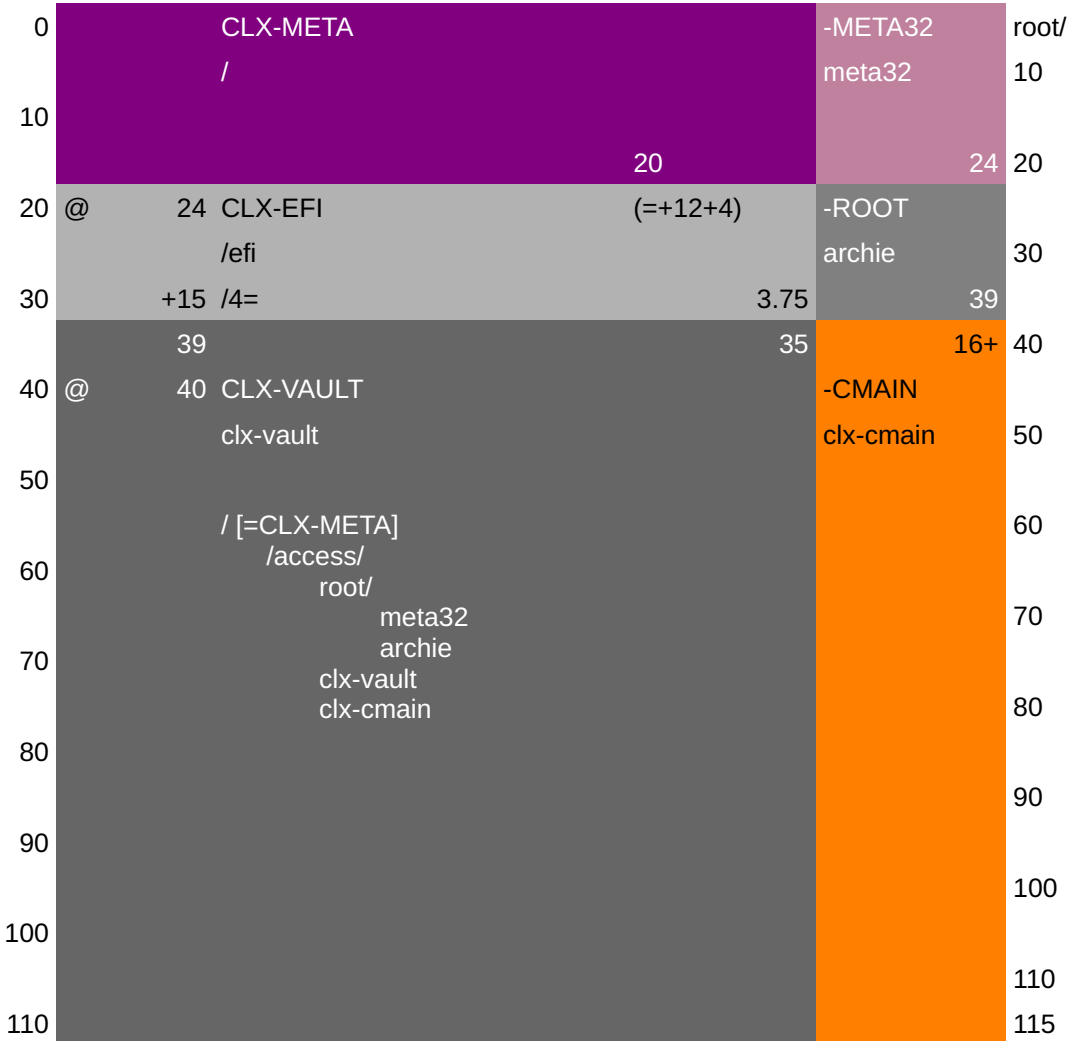
//==Crystals Linux Setup=====

// [00 – Overview]

/:

Preliminary: it is suggested that you either acquire a ‘good’ (128 sGB, 300-400 sMB/s minimum) flash drive, or a proper external hard-drive; however: any will do. It is however to be mentioned, that repeated read/write operations reduce the lifespan of a flash drive; While it is said that modern devices don’t suffer from this problem as much.

Furthermore: the use of an external setup allows us to more uniformly engage with the system. It can be experimental and unique in its design, without tempering too much with an existing system. To this extent the identifier CLX is introduced as alias for the whole. It is designed to externally dock with an existing system, while being its own independent development and metastorage device. The following setup will be implied throughout this document.



Additionally: it is supposed that the user is a Linux noob; Yet (and/or therefore) this is implicitly an Arch Linux installation “at minimum”. The rationale is as follows:

1. We want a simple and minimalistic setup that we can comprehensively work with.
2. The motto is “You do You” – to which the system highlights as a commonly shared node through which we can unite with respect to our individuality.

Respectively, as a guideline: We avoid installing qt and xorg. Any package that has a dependency on a desktop (i.e. GNOME or plasma) is disqualified. We use the gtk desktop portal and intend to leave it at that.

Importantly: I opted to **use btrfs** – and suggest you to do the same.

1. As it is lighter on read/write operations, it is definitely the smarter choice for a flash drive.
2. Having subvolumes is the kind of gadget we want.
3. I treat it as an experiment as I cannot really tell what and why.

[see ArchLinux.org > Install on a Removable Medium](#)

Note: Eventually this is supposed to be(come) a multiboot environment; If need be 'with' the capability to boot into a 32bit system. To that end I recommend also reading the articles:

> USB flash installation medium

> Multiboot USB drive

> Install Arch Linux from existing Linux

meta32 and archie are meant to mirror the Arch Linux 32 and Arch Linux live iso's while somewhere I'm sure we also have space for chainloading raw iso files. I would however not recommend using this drive as a general purpose "distro cruiser".

Importantly: watch out for the remarks on grub installation and kernel hooks.

NOOB Corner:

What any of this means: It is relatively easy to get used to and come to terms with the various peculiarities of using Linux for personal computing. Doing so will inherently teach you some very basic things about computers, while different distributions have different knowledge requirements for the end user. The deeper you go, the more complex things become; And the more you yet again rely on the work of others to get certain things done.

Any distribution can serve as a Lens into the intricacies of what Linux is at large – and also into the inner workings of a computer; Albeit: with the Unix/Linux spin to it.

Arch Linux allows us to spin up a really basic Linux System – and to understand what I mean by that, we need to understand something about "package philosophy". Package Philosophy is to say that we all have an idea of how software is supposed to be packaged for distribution; Or conversely how software is to be installed. This would usually just integrate the software into the Operating System, counter to which the "plug and play" concept exists.

Package Philosophy, for the developers, eventually entails more than just the package itself. Global or Common resources are an essential and non-negotiable aspect of how Operating Systems function – and subsequently "packaging tools" are born to manage and automate these processes.

Linux Repositories are, as the name suggests, repositories of individual packages. The purpose of a package manager is ultimately to install any package you want, while also installing its 'dependencies'. A distribution's package manager is therefore pivotal to what it (the distribution) tries to be. Given that we all mostly care for the same 'software', the difference is usually only a matter of if and how a given package is available. But while a generic "Desktop Distribution" would almost implicitly install xorg, a display manager and a desktop environment – along with a few things that it wants in order to supply a smart and easy to use product, in Arch Linux we can totally do without any of that. So, while Ubuntu or Mint would care to integrate packages into the Desktop experience – in Arch we usually just download the respective packages while we have to set them up ourselves. So, while simply installing plasma and letting pacman do the rest is possible, one still has to setup their system; Whether or what display manager to use, etc..

About Crystals Linux and Wayland: Back in the 2010s, when I first started using Linux, things were a bit different; Although now, 15 years later, things pretty much seemed the way they used to, at first. However, installing Arch has gotten easier; And I assume that the same is true for Gentoo. Trying Gentoo would be the next logical step – assuming an even deeper level of control – but to that end it would first be prudent to understand the basics.

And so does this history dive first of all take us to this little thing called a Framebuffer.

It is certainly not just me who had a somewhat computerphilic attraction to it – within the context of Linux – quickly letting go of anything that implicitly relied on xorg while trying to expand the linux shell itself into the modern age. And thus Linux Porn – as they call it – was born. See: **fbterm**, **tmux**.

So I wondered, now, 15 years later, what developments have been made. And it would seem that the most of it is "AUR walled". The rationale from our point of view however is clear: Why not try to develop the Crystals Meta as a Window Manager/Desktop Environment from the get go? (as from first principles, "a.k.a." STANDALONE)

An argument I stumbled upon in a Forum about 'why' these gadgets aren't in the official repos (?) is that ultimately they're just gimmicks. For any real use you'd want access to the programs you need and subsequently you need the one thing which then needs another thing and so you're back to square one.

It took me a while then, to actually understand what Wayland is. And thereafter it is pretty much implied that our Desktop of Choice is sway. Sway is tightly involved with wlroots – which explains itself as: (the) thousands of line of code you'd write anyway – referring to wayland and its utility as a low level framework for your computer's resources. On top of that is Sway functional and simple – said to be the one that's the most fleshed out. Whether or not I want to use wlroots will come down to whether or not it can live up to its claim. But for the start I think it's best to ignore it. *Instead* I fetched woodland from the AUR.

From what I gather, a minimalistic xorg WM would be easier to produce (less code) – at least at the beginning. But since I don't intend to use any xorg functionality within my project (just yet), that isn't a priority right now.

System Pitch:

So: a 'basic' Linux system here at first means: A functional shell. Then we add Sway. If you're familiar with i3, this is that but for Wayland; And both have rightfully claimed their spot as the best Linux WM's out there. I don't fully understand how Wayland does what it does – I'm yet to dive into it – but in essence it is the modern replacement for fbterm. But rather than displaying a terminal within itself – it relies on a 'compositor' to make anything happen. Crystals follows a similar concept. Respectively Wayland can be seen as System Core, whereas the Compositor is equivalent to the Meta. Maybe this isn't quite right, but in simple terms.

To get to what fbterm does or did, all we need is a compositor and a terminal emulator. Configuring Sway to automatically start foot at launch is basically that. It has on-board integration for an execution prompt, workspaces, status monitors and desktop programs. For reference: I only installed libreoffice and copied the sublime-text installation from my system. Both didn't work out of the box. After installing gtk3 sublime-text would at least start, libreoffice however complained. Installing xdg-desktop-portal-gtk fixed it (See: Arch Linux – XDG Desktop Portal). So: xdg-desktop-portal-gtk (uses gtk3) is a very basic "desktop backend" that various programs rely on – mainly: the file chooser portal, I argue. So, without it – there's nothing to browse the filesystem with. While the gtk portal is marked as 'generic', it also comes without an entire desktop environment attached to itself. And that's what I call **minimalistic and lightweight**, while also keeping it Simple stupid.

Well, all in all the installation entails 4,4 GB – 5,5 GB marked as allocated (- 1,1 GB cached).

Arch Linux Installation: To tie things back together, do for now file the previous segment (think: framebuffer) under 'Kernel Compilation' – as for now I want to use the Arch Linux Installation as a prism for what's going on at first.

1st Note: The way the installation on a flash medium is performed is virtually identical to a normal installation. You can very much create an installation medium to install Arch on a USB drive (as 'from an existing Arch Linux installation').

The first step at any point would be to prepare the Storage Device. But specific to Arch, the first step is to configure the Live Environment to connect with the Internet so it can access the Arch Repositories. Then a specialized program is used to download the 'base' package into the "root to be", at which point the root is also being created.

/etc	(editable text config)	global configuration files
/home		user roots
/root		root user root
/boot, /efi		boot config, kernel(s)
/dev		device sockets(?)
and everything else is in some way established practice and largely beyond me.		
/usr/share		entails icons and themes
/usr/bin		entails executables

Important at first (after installing the corresponding packages) are

/boot/grub/grub.conf	kernel parameters
/etc/default/grub	grub configuration (splash image)
/etc/mkinitcpio.conf	kernel hooks
/etc/fstab	mount configurations

I found 'nano' to be the least confusing editor to use. Many mention 'vim' – and the key issue I suppose is muscle memory. Using nano and mc side by side can lead to problems. Ctrl+o in nano is save, while ctrl+o in mc toggles between browser and shell. When then nano-ing a file within the mc shell and pressing ctrl+o will create a bit of a mess. So, to understand vim is to understand more than just vim. Not knowing vim – is knowing less than not only vim. But alas, one does not have to know vim. Then google: "how to [...] vim"

So, after 'installing' the base package to the target root, we 'arch-chroot' into it, saying: the target path is now the root of a virtual environment and the system behaves as though that root were its root. However: as part of the arch-chroot we also take some aspects of the live system into the new root environment. Most importantly: the internet connection. You could alternately try to install iwdb along with base and configure the rest from within a normal root. It is at this point also handy to have any old iwdb config files ready, namely

```
/etc/iwdb/main.conf
/var/lib/iwdb/yournetwork.psk
```

Most problems I encountered during this stage is with not understanding systemd and which parts of this process rely on it for what reason. Here a short breakdown:

systemd manages daemons. These daemons are configured through service files. For an example try to look up how to enable numlock on boot and check out the method that utilizes one (though this method is imperfect. Some programs still read the numlock as off).

What is a daemon you ask? Generally described as a background process, there are a variety of reasons why one would use them and hence why they are effectively necessary. The key term is 'IPC' or 'Inter Process Communication'. And here the term "daemon" alias 'background process' is also a little bit misleading. For example:

iwdb requires the "systemd-networkd.service" for networking. Iwctl won't work if it hasn't been enabled. What exactly is going on internally I do not know. But looking back to the part about packages: A part of the package ecosystem are libraries. Individual programs can use these shared objects as though they were a part of themselves; But two programs that share one such object still hold individual copies of it. If you want both programs to use a single shared copy; That single shared copy needs to somehow exist as its own thing – and the system needs to be able to address it. So, in all simplicity a daemon can be as little as a piece of memory; Alongside some set of functions to manage it.

iwdb further requires the "systemd-resolved.service" in order to resolve namespaces. That is, to translate the machine speak of the interweb into the words we use to navigate it. Whether or not this is the best solution I'm not sure about. The two aforementioned ones alongside the "iwdb.service" could very well be combined into one – whereas the ability to customize its scope via the use of daemons lends itself to a broader scope of utility (more function, less bloat) which, by the rule of adaptation, should in turn increase its stability and security.

→ See: Arch Reference Manual : CORE PACKAGES.

So, once networking is in order (though to be sure you'll have to reboot into the new environment) the next step is to set things up. Or maybe this is the first thing one would do. The core packages entail the implied minimum, base-devel (which includes gcc) (and/)for git (git to interact with the AUR), basic filesystem tools, and some basic essentials (nano=editor, mc=browser). *consider vim.

It is the setup to have a setup for to set things up with.

It is then at this point that you can configure anything at all – such as your locale and vconsole settings. Those are for some reason required by mkinitcpio to compile the kernel – which is for now and here a neat little quirk to shift some more attention on it:

I however have, for all the time I've been using linux, never had to really mess with it; Outside of applying kernel parameters in the bootloader and messing around with the HOOKs as instructed. And about kernel parameters in the bootloader: I like to disable the 'quiet' setting. I tried to get more intimate with them, but to this point I think I cannot safely tell you what either one of those two files is in particular – like, which one is which. Hmm OK, is it so: One is the kernel (usually vmlinuz) and the other is the "ram FS" (usually ramfs) The kernel is itself the kernel that is being loaded, whereas ramfs happens to be an initial filesystem or resource pool for the kernel to draw from during startup. The bootloader has the purpose of knowing where to find them – while being itself the program that is initialized by the hardware.

So, at occasion some processes require the kernel to be recompiled. That's the purpose of mkinitcpio.

... And by editing your hostname you can give your root a name. Finally you'd create a user and setup passwords. Creating a user is important as it is expressed best practice to not interact with the AUR as root. In as far as Crystals Linux is concerned we have our individual type of root – and after setting it up, it is 'safe' to reboot – check if everything works – before moving on with the rest. If you know what you're doing you'd just finish up and check then.

→ See: this : 01 – Environment

Et voila: A basic, functional and minimalistic Linux Shell.

NOTE: This setup comes without PDF and Browsing capabilities. I haven't figured that part out yet.

You Do You – Simplicity:

You Do You is here code for: For people that might as well still be using Windows. We might as well refer to them as "Plebs" – not merely by choice but by virtue. Yet is there an intersection – a transitional boundary perhaps – that is ultimately however more so a circle in and of itself. Ye who come and are thirsty, ye seek water. Ye who come and are hungry, ye seek food. But what did ye find?

```
//==Crystals Linux Setup=====
// [01 – Environment]
```

1: users and groups

```
#-----
# import skeleton:      /etc/skel
                        .config/
                        sway/...
                        foot/...
                        fbterm/...
                        .bash_profile
                        .bash_rc
                        .i3status.conf
# clx-setup/01-00_skel/
#-----

useradd -m meta
useradd -mU -u 2000 cmeta
groupadd -g 1200 -U meta,cmeta      cmain
```

```

groupadd -g 1201 -U meta,cmeta      circle
groupadd -g 2200 -U meta,cmeta      meta_user

# import meta key:      /home/meta/
                             .box/
                             i3config/..
                             issue
                             issue.systag
                             issue.idtag
                             test_issue
                             .bash_profile
                             profile.public
                             profile.meta
                             .bashrc
                             Desktop/main
                             #see: meta_profile.readme
                             /root/
                             .bash_profile
                             .bash_rc
                             .rootrc
                             command/
                             mount.home
                             mount.dock
                             umount.home
                             umount.dock
                             ...
                             #see: command.readme
                             /etc/
                             profile.d/
                             00-clx_header.sh #empty
                             91-clx_profile.sh #empty
                             issue
                             dircolors

# clx-setup/01-01_meta/
#-----

usermod -aG wheel meta
usermod -aG wheel cmeta

#-----

```

NOOB Corner:

What is Crystals Linux? In the first phase of the environment setup, the focus is on setting up the meta user; Which is basically the root user within the Crystals ecosystem. It is suggested that this is the first user and subsequently first user-group of the system (ID=1000), saying → it is presumed that the first user of a system is the meta root of that system. It encourages a sense of etiquette. Since this collides with the likelihood of the meta of a system going by a different id, we might pool around an alternative (2000); But consider, what user account you intend to use in order to interact with the code. Assuming that it is your computer and you are using your primary account, your files will be marked with your user-id. Hence: meta uid=1000 on the flash drive will implicitly mirror your system's primary user, speaking of the filesystem footprint. Hereto, conceptually there are three aspects to this:

- 1: System Side Development
- 2: User Side Development
- 3: End User/Adaptive Development

Drawing upon some homebrew philosophy, Linux typically sets you off at 1000 – and unless you're specific each user and group will simply pile up. Also note that user id and group id aren't the same. A user's group id comes in

form of an individual group id that can differ from the user id. So, being careful can cause chaos and confusion. But reduced to its principality, the “core” user of a Linux system is uid=1000. That is an implicit wildcard – mirroring the diversity entailed therein. Within a network however, this user is implicitly anything ‘but’ that, let it be an abstract, common core it is implicitly a part/child of. From one’s self to “the other” – a consequence of identity is the abstract, a translation of one’s self into a unique expression.

On a different note then, each of these three aspects corresponds to a different degree of openness one would want from their system. Aspect 1 wants/needs it all to be open, aspect 2 needs it all to be locked but accessible still – plus some extra space to expand within and aspect 3 naturally wants it all to be locked and wrapped into an end-user experience.

To wrap this into a definitive statement of sorts, here’s what this means:

1. It is a dedicated environment to develop Crystals in its ‘open’ form that inherently ties into the user’s own system (→ root and meta integration).
2. We assume liberty to determine what a “stage 2” and a “stage 3” system are to look like, and take advantage of the fact that we have root access on a flash drive.

Or in other words: The flash environment serves as an inflection point that on one end adapts to an existing environment such as to manage a crystals installation thereon, primarily focusing on the integration of personal data, while being able to also to produce a new standard that can then be packaged into a stage 2 environment.

The ‘meta-key’ “completes” the setup stored in /etc/skel, containing a ‘meta profile’ that sets up a few environment variables that we use to streamline our scripts. Thus introduced by the meta user, who has access to the meta profile, which is key to unlocking the crystals root.

Upon invocation the script tests if it can access /root/.rootrc – and if so also invokes that. An echo then indicates whether it initialized as meta or as root. The .rootrc entails aliases that perform root operations. Specifically rgo and rgo_command – which are there to cd into /root and /root/command respectively. Root command is to contain scripts that perform repetitive system operations such as mounting and dismounting filesystems – but also to store and restore system settings. Profile files can be used to specify paths and a script will copy them between specified locations.

Parallel to that there is ~/Desktop/main – which shall be the place from where we interact with the AUR and stuff. Again counter to that there is ~/.box, alias a container for our scripting adventures.

- > dock refers to auxiliary volumes on a system and are mounted to /home/dock
- > home refers to a system’s root. The root is mounted to /access/root/system

The “CLX Gate”: Is for all intents and purposes a crystals specific mount folder. It is hereby presumed that your individual preferences are wildly unconventional or absolutely not within the scope of simple serialization – such that it is prudent to find ways towards common grounds. A first instance thereof is language. Written into the meta profile are paths, that serve as ‘bridge of privilege’ – constituting a basis of integrity between the different “nodes” independent of device-state. Respectively cmeta is there for when the meta environment is inappropriate.

But first:

2: crystals

```
#-----  
#  
> CLX-CMAIN:  
  
# create ‘source’ subvolume  
  
cd /access/clx-cmain  
btrfs subvolume create source
```

```

# create main directories
/access/clx-cmain/
                                crystals/  #'active'(primary) source
                                public/     #shared objects, executables
                                _Docs/      #drop-ins

# link home

cd ~
ln - /access/clx-main _cmain

> CLX-VAULT:

# create subvolumes

btrfs subvolume create          backups
                                documents

# link home

cd ~
ln - /access/clx-vault/documents Documents

# create clx gate

as root:
mkdir /clx                      #a public gate managed by root
chown -R meta:cmain             /access/clx-cmain
...
~ according to preference ~

#
#-----
# Where:

# you can then symlink folders from your dock into
# your documents folder – and have them easily accessible
# from within your flash drive.

```

```

//==Crystals Linux Setup=====
// [02 – crompt]
/:

```

Installation in Flash Environment: It is to be noted that this is ultimately the system through which I can communicate my progress in a streamlined fashion. If you find yourself at this point and you create your own, your first dependency would or should be me. You would begin by downloading some packages and files – and store them in “\$CLX_MAIN/source/import” – where you can browse and manage your installation with midnight commander. To first get started, you want to make sure that crompt and the crystals folder are where you want them to be. The official target for the cpp folder is clx-main/crystals and the official target for crompt is clx-main/public. Then

```
ln -s /path/to/crompt /usr/bin/crompt
```

and adjust its parameters until it works. The CLX-ROOT_PACKAGE's package container contains a simple cpp file to use for testing. My practice is to compile it from ~/ljack.cpp into ~/jack. Once you get a successful compile, you can consider this task completed. Before moving on, here a few notes:

"Let crompt as such" be the way to compile any source that is passed to it through the active installation and a dedicated project source (crompt-source). This 'primary source' denotes an active meta, whereas the current goal is to develop crystals into three conceptually distinct systems. The primary project is to establish crystals as a wayland compositor with an on board 3D-Engine/VR. The secondary goal is to establish crystals as a daemon that integrates with an environment it is operating within. The third goal is to establish crystals as a framework for the development of "Crystals" (singular: a crystal) – of which there are two forms: Super Crystals are monolithic executables that would run as individual programs for instance on other operating systems or within other DEs/WMs. Basic Crystals are binaries that make use of an active crystals daemon. Pure Crystals are thought of as Crystals that function within the "primary (native) domain", Aligned Crystals are thought of as Crystals that align with an external System (i.e. xorg, network manager, etc.).

At this point in time, the current official version is release 5, with my version being modified in accordance to the stated guidelines. From there, let crompt be what crompt is – which is to produce a basic linux executable that can be run as a standalone program. It is to declare itself as either __SUPER_CRYSTAL__ "super-id" or __CRYSTAL__ "crystal-id". It is the basis from which we move forward.

- The primary goal requires us to create a new System Core that integrates with Wayland.
- The secondary goal requires us to create a new System Component that integrates with Linux.
- The third goal requires us to flesh out System Utility utilizing SDL and OpenGL.

Challenges: The new System Core does at first glance, conceptually, not differ by much from the old one. The practical difference is to first initialize the Wayland backend, and to initialize SDL on top of them. In practicality it is however, first and foremost, its own thing that has to work to some extent before integrating CrystalsGL.

The question however becomes how we want to treat the M-Core in this design. A sensible solution would be to create a new branch to further highlight, isolate and adapt-to the significant changes. On a first thought then it seems, that the cmain is the pivotal carrier of this new branch.

```
ln -s /path/to/cryme /usr/bin/cryme
```

As crompt, cryme utilizes the active source. Instead of being given access to the crompt-source it is given access to metasys; And is at its heart identical to calling crompt from within /crystals/metasys. Using crompt, the crompt-source is being accessed, whereas cryme can be used to step outside of it.

...