NTOScript User Manual

A Forth scripting environment for INDI devices.

Rafael Gonzalez Fuentetaja

NTOScript
User manual
Manual revision 1.00
12 July 2008

Table of Contents

1 Introduction

1.1 Audience

This manual is aimed at application programmers who wish to write INDI control scripts. A more advance profile would also include those programmers that wish to write GUI front-ends using the INDI protocol through `NTOScript`.

Both control scripts and `NTOScript` itself are written in the `FORTH` language. This manual is **not** a FORTH Primer, although some rationale behind using FORTH is given. If you are new to FORTH, skim first through this manual and then refer to the appropiate material given in

Chapter 1 [Reference documents], page 1.

This manual does not analyze in detail the `INDI` protocol. For a complete reference of `INDI`, please refer to the proper link in

Chapter 1 [Reference documents], page 1.

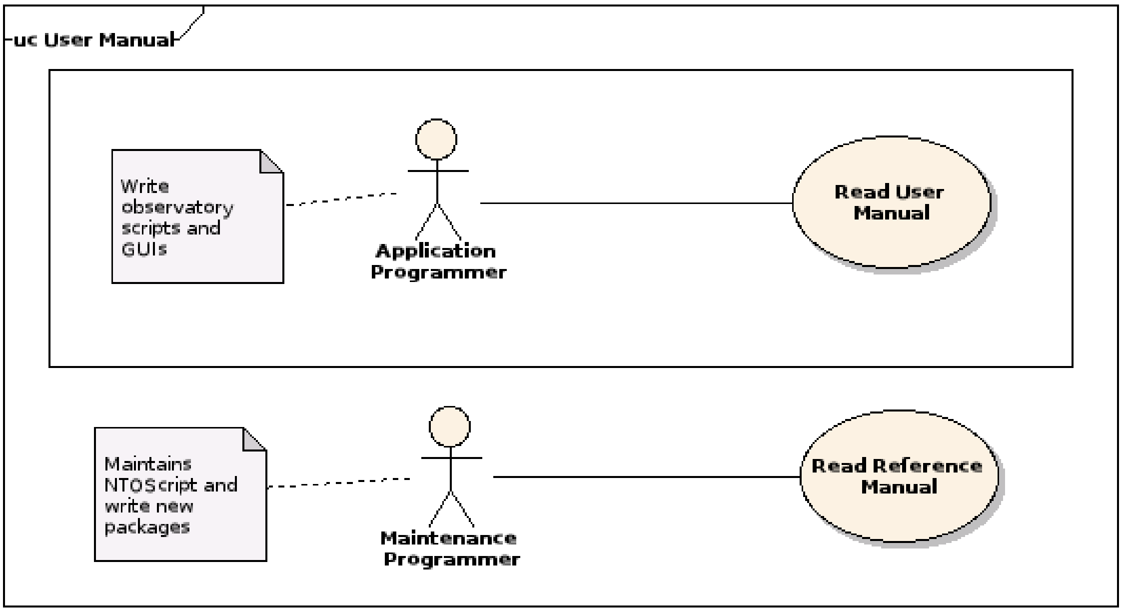

Figure 1.1: User Manual audience.

1.2 Reference documents

INDI and FORTH material.

1. INDI: Instrument-Neutral Distributed Interface v1.7. The Reference specification, owned and maintained by The ClearSky Institute
2. The INDI SourceForge Project is the official site for everything else but the above specification.
3. *VFX Forth for Linux User Manual* as the definitive reference for maintaining and expanding NTOScript. The evaluation license for Linux is free. Visit MPE Website for details.
4. Programming Forth, by Stephen Pelc. A modern FORTH tutorial based on VFX Forth.
5. Starting Forth (on-line ed.) at Forth Inc.'s website. For some, the best introductory book on FORTH ever written.

6. Another classical reference is *Forth Programmer's Handbook* by E. K. Conklin and E.D. Rather, distributed with the SwiftForth free evaluation licenses, also at Forth Inc.'s website.
7. And so Forth, by Hans Bezemer.
8. Introduccion a Forth (Spanish) by Javier Gil Chica.

2 INDI in a nutshell

INDI stands for Instrument Neutral Distributed Interface. This specification was first released by Elwood Charles Downey of the ClearSky Institute, on 2003. It is copyrighted and maintained by him. The current protocol version, as the date of writting this manual, is 1.7.

Unlike other methods to control astronomical devices[1], this one is based on neutral technology: XML messages. Other features of INDI are the use of a client/server paradigm and a stateless protocol.

In practice, INDI clients do not communicate directly with the devices. Instead, they contact an *indiserver*, a mediator between clients and devices that can offer additional services such as message routing and authentication.

2.1 INDI properties

In the INDI world, each device present to the client software a series of visible *Properties*. Properties are really *Property vectors*, arrays of one or more components or *Property items*. This is a very convenient feature.

Imagine a telescope `Mount` device whose property `EQUATORIAL_COORD` describes the `RA` and `DEC` positions. The client software fills each component separately. Once the individual `EQUATORIAL_COORD` components have been set, you can send the whole property vector to the device, which will start slewing to that position.

```
00:50:00 R.A.   property item   Mount EQUATORIAL_COORD RA  set
39:46:00 DEC.   property item   Mount EQUATORIAL_COORD DEC set
                property vector Mount EQUATORIAL_COORD send
```

The INDI specification describes a series of XML messages and rules, to communicate property values from client to server and vice-versa. In the example above, the typed commands will cause the following message to be sent from the client software to the `Mount` device:

```
<newNumberVector name='EQUATORIAL_COORD' device='Mount'>
  <oneNumber name='RA'>1.2500000e1 </oneNumber>
  <oneNumber name='DEC'>3.9766666e1 </oneNumber>
</newNumberVector>
```

The device might respond right away with something like this:

```
<setNumberVector device="Mount" name="EQUATORIAL_COORD" state="Busy"
 timeout="36" timestamp="2008-03-30T09:14:54"
 message="Slewing to RA 12:30:00.0 Dec  39:46:00">
   <oneNumber name="RA">
      12.50000023400770
   </oneNumber>
   <oneNumber name="DEC">
      39.7666659139457
   </oneNumber>
</setNumberVector>
```

And when the slew is complete, the device might send a new message like:

[1] Most notably ASCOM, tied to Windows technology

```
<setNumberVector device="Mount" name="EQUATORIAL_COORD" state="Ok"
 timeout="36" timestamp="2008-03-30T09:14:54" message="Now tracking">
   <oneNumber name="RA">
      12.50000023400770
   </oneNumber>
   <oneNumber name="DEC">
      39.76666591394570
   </oneNumber>
</setNumberVector>
```

All communication is asynchronous. The client software send new values for the property vectors and may or may not wait for answers. This is particulary true for GUI clients, which do not wait for responses. Instead, they can act upon INDI messages as soon as they arrive.

2.2 INDI property states

In case you have not noticed from the XML messages above, property vectors as a whole have a *state*, which gives us a hint on what's going on in the device. The device can be in either one of the following states: `Idle`, `Ok`, `Busy` or `Alert`.

In the example above, the property `EQUATORIAL_COORD` goes into a `Busy` state until the `Mount` device reaches the target equatorial position. At that time, the `EQUATORIAL_COORD` gets back to the `Ok` state. At any time, a property can enter into the the `Alert` state. For instance, the `Mount` above hit one microswitch that controls the slew limit.

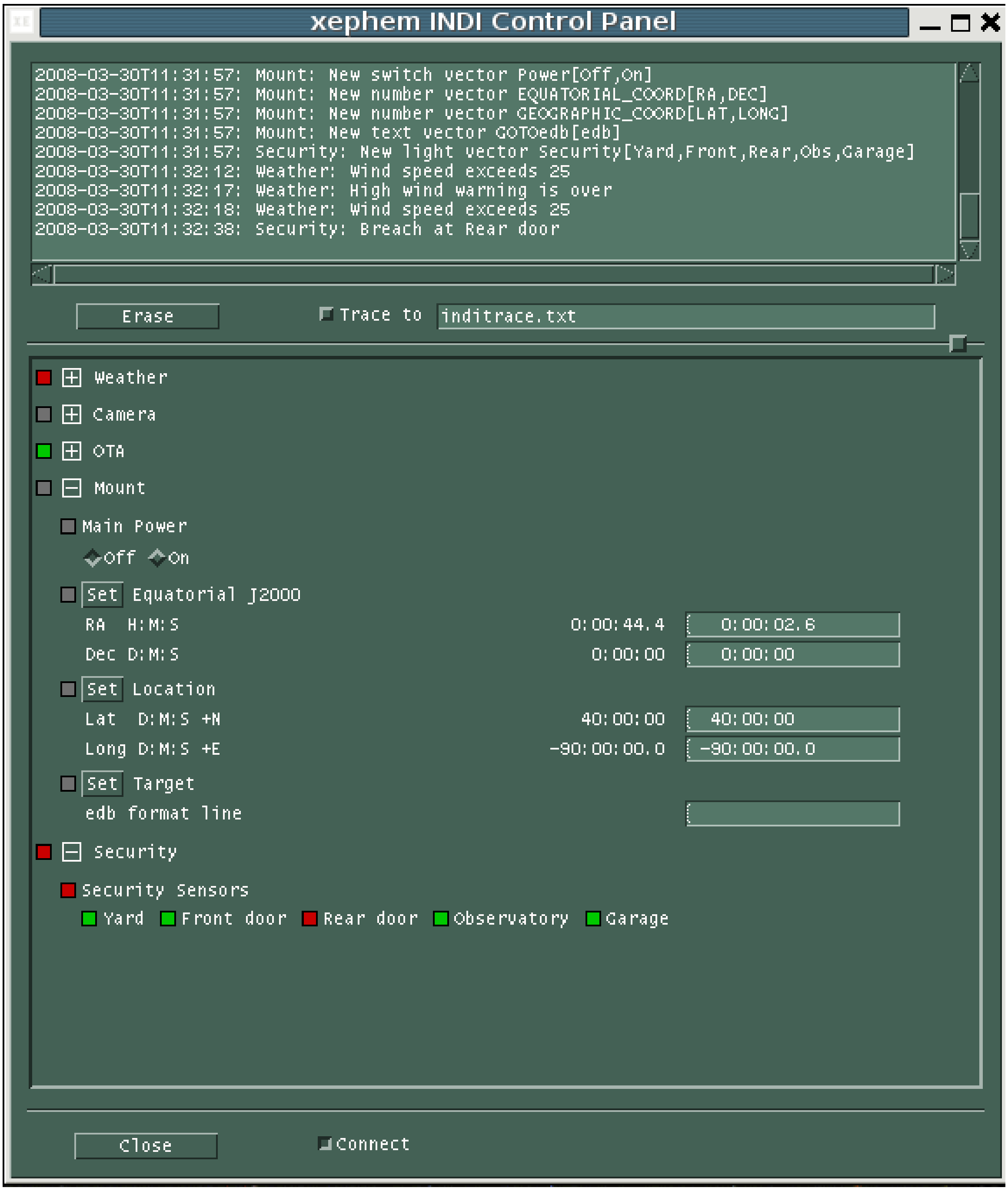

Figure 2.1: XEphem's INDI Control panel.

GUIs can visually represent state as colors, as seen on Figure 2.1. The suggested colors are gray for `Idle`, green for `Ok`, yellow for `Busy` and red for `Alarm`.

2.3 Property vector types

Under INDI, property vectors are classified as:

1. *Text* property vectors. Text property vectors hold text items like comments or free text you want to enter. Property *Target* on Figure 2.1 show a single text entry box for its only component *edb format line*.
2. *Number* property vectors. These ones hold arrays of numbers. These property vectors have

other features such as the min and max allowed values or the step increment. Property *Equatorial J2000* on Figure 2.1 show this property vector with two components, *RA* and *Dec* with their respective entry boxes.

3. *Switch* property vectors. These represent things that can be switched on or off, options or trigger actions. Property *Main Power* in Figure 2.1 shosw the state of the AC mains power supply and how can be turned on or off.
4. *Light* property vectors. Light vectors represent arrays or read-only lights that can signal certain events, like the *Security Sensors* property on Figure 2.1. Light values are `Idle`, `Ok`, `Busy` or `Alert`.
5. *BLOB* property vectors. These represent binary data, such as images, that cannot be handled using text vectors. They are usually sent from the device to the client software, although INDI allows sending BLOBs to the device for exotic purposes like updating a device's firmware. Not all clients can handle BLOBs, so BLOB reception is disabled by default and the client software must explicitely enable reception.

2.4 Discovering INDI devices and properties

Other amateur astronomy automation technologies define a common but minimum set of features, grouped in interfaces, that all astronomical devices must abide. Under INDI, each device may have its own personality and announces itself to the rest of the world.

This dynamic description and run-time discovery is mostly convenient for GUIs, which can avoid hard-coding the interface elements for a certain device, making them in fact **universal and adaptive** controllers.

XEphem's INDI Control Panel in Figure 2.1 is just an example of an universal, adaptive INDI GUI. Of course, nothing prevents a developer to write a custom, hard-coded GUI for an specific device, knowing beforehand the amount and type of property vectors this particular device exhibits.

Dynamic discovery is less an advantage for scripting, since the application programmer **must** have "a priori" knowledge of devices, properties **and its behaviour** to write the script.

Some effort on standarizing properties has been made. See [INDI reference material], page 1

3 NTOScript Tutorial

3.1 Why FORTH ?

FORTH is an old language, certainly older that the popular mainstream languages available nowadays (Java, Python, Perl, Tcl, Visual Basic, Ruby, C++,etc).

It enjoyed a golden age around the 1980 decade. Now it can be found in specific markets like embedded systems, booting workstations, etc. It is definitely **not** mainstream anymore. Ironically, the first application field for FORTH was telescope control, (*The FORTH Program for Spectral Line Observing, C. H. Moore and E. Rather, IEEE Proceedings, Vol 61, No. 9, September 1973*).

Excelent tutorials on FORTH are available on Chapter 1 [Reference documents], page 1. The one by [Stephen Pelc], page 1 even addresses the controversial aspects of FORTH in chapter *Adopting and managing FORTH*.

So, what's the purpose of developing a control environment in FORTH ?

- Above all, a matter of personal preference. **FORTH is fun to program**.
- Like other well known scripting languages, **you do not need any IDE**, just an editor and a console.
- Like other well known scripting languages, **FORTH is dynamically typed**. No need to specify types, nor function or procedure prototypes.
- Like other well known scripting languages, **FORTH is interpreted**. This allows you a very quick edit/test turnaround cycle.
- Like other well known systems languages, like Java or C/C++, **FORTH is also compiled**. Other scripting languages also compile to an intermediate code to improve performance. FORTH typically outperforms these scripting languages. Modern FORTH compilers like VFX produce optimized machine code that can compare with C performance.

  I have been trying telescope control with a couple of scripting languages. Yes, they are powerful, easy to program, full of libraries and even you can write GUIs with them. But after developping a good amount of software, guess what ... it is slow ! There are usually workarounds like writting in C the critical parts, loosing your interacivity. Another one is using external utilities that "compile" your software and produce a bloated version. I also prefer efficiency in code space.
- **FORTH has a neat syntax: only words separated by spaces**. This results in a much clearer syntax that is very readable, if properly designed. No parenthesis nor dots, no long names if possible. The example in Chapter 3 [INDI in a nutshell], page 7 is written in NTOScript.
- **Modern FORTHs can access external, shared libaries** if needed. I certainy appreciate not to reinvent the wheel whenever possible. VFX is particulary friendly in this aspect.

In summary, *the only language I have found that is both interpreted and compiled, offering at the same time a quick edit/test cycle, superb performance, easy interfacing capabilities and a neat syntax is FORTH.*

3.2 Invoking NTOScript

If properly installed, NTOScript should be invoked by just typing:

```
ntoscript
```

Exit from NTOScript by typing:

```
bye
```

Executing backgorund jobs

If you wan to execute a given script as a background job to cron or similar facility, NTOScript should be invoked as:

```
ntoscript include /<path>/<to>/<your>/<script>
```

3.3 Script programming cookbook

This section introduces you to the basics of programming in NTOScript. By now, you should have clear that a FORTH program consists of a series of *words* sperated by spaces and each word have a role to play in the program. We use an informal, cookbook approach that shows many examples with prototype phrases.

All examples given in this cookbook are real examples you can play with. They can can be found at '`/usr/local/ntoscript/scripts`'. The supporting software is the excellent `XEphem` sky-charting software. Download the whole tarball from the Clear Sky Institute website, compile both xephem and the '`tools/indi`' directory[1].

Skim through this cookbook to get a first impression. Then, take some time getting started to FORTH (see Chapter 1 [Reference documents], page 1). A second reading should make things more self-evident and you will be ready to have a look at the reference section (Chapter 4 [Properties database], page 35 through Chapter 8 [Sexagesimal Numbers], page 69).

The first use case to be developed is a simple CCD imaging session of M31 galaxy through various filters and CCD camera, attached to an OTA supported by a robotic equatorial mount. The OTA itself includes a focuser and a filter wheel. A weather station monitors humidity and wind speed.

Example code will be shown in rounded rectangles or "cartouches" with one or several discussion topics after each example.

Starting up the server

After having compiled `xephem`, go to the '`GUI/xephem/tools/indi`' directory and type `make`. The following driver are compiled:

- '`indiserver`'. Server program managing drivers below.
- '`cam`'. A CCD camera simulator.
- '`tmount`'. A Telescope mount simulator.
- '`ota`'. The Optical Tube Assembly simulator.
- '`wx`'. A Weather station simulator.
- '`security`'. A security system simulator (we won't use it).

Then, type in a separate terminal window:

```
./indiserver -v ./tmount ./cam ./ota ./wx
```

to start the remote INDI server.

3.3.1 Core package programming

The *Core package* is the part of NTOScript that deals with basic INDI device control.

---

[1] Unfortunately, the supplied library file '`indidrivermain.c`' formats floating point numbers with too many decimal digits and breaks the FORTH input conversion routine. `C`'s 'double' floating point precision is only 16 digits. All occurences of %.20g in this file should be replaced by %.16g.

3.3.1.1 Connection and device discovery

Example 1

As said in Chapter 3 [INDI in a nutshell], page 7, we need first to connect to a remote INDI server to control INDI devices.

```
IndiServer: indiserver localhost 7624
indiserver connect
```

Rationale: Connection to an INDI server

The above phrases declare and creates an object named `indiserver` that will connectto a given host and TCP port (`localhost` and `7624` in this case). After this object has been created, we issue the `connect` command on it.

If we have not started our '`indiserver`' process in another window, this is what happens:

```
Err# 111 Connection refused
 -> connect
```

By the way, there is also the possibility for you to disconnect from the remote '`indiserver`' process by issuing:

```
indiserver disconnect
```

Example 2

After the connection is done, the next thing to do is to discover remote devices. We do that with the following phrases:

```
indiserver properties get
2 seconds waiting
```

Rationale

For interactive use, the final wait is not necessary, because the discovery process is fast compared to our human reaction time. However, it is necessary for scripts, since the next thing to do will be to get and set property items which may not be known to NTOScript by the time your script execute them.

Example 3

For interactive console use, you may print what devices and properties we have discovered by issuing:

```
indiserver print
```

and, depending on what devices we have started on the remote side, we might get something like this:

```
Device Weather

Property WX (Ok,ro,60 ) [Weather Station] []
  Pres = 9.9988358e2 +0 +0 +0 %6.1f [Pressure, hPa]
  Temp = 9.9773267 +0 +0 +0 %6.1f [Air temp, C]
  DewP = 6.0658814 +0 +0 +0 %6.1f [Dew point, C]
  WDir = 9.2274145e1 +0 +0 +0 %6.1f [Wind dir, EofN]
  WSpd = 2.3858043e1 +0 +0 +0 %6.1f [Wind speed, kph]

Property WXAlert (Idle,rw,60 ) [Weather alert settings] []
  MaxDewP = -3 -1000 +0 +1 %6.1f [Dew limit from temp]
  MaxWndS = +25 +0 +1000 +1 %6.1f [Wind speed limit]


=======================================================================
Device Mount

Property Power (Idle,rw,60 ) OneOfMany [Main Power] []
  Off =  On [Off]
  On =  Off [On]

Property EQUATORIAL_COORD (Idle,rw,36 ) [Equatorial J2000] []
  RA = 3.8638666e-3 +0 +24 +0 %12.8m [RA  H:M:S]
  DEC = +0 -90 +90 +0 %10.6m [Dec D:M:S]

Property GEOGRAPHIC_COORD (Idle,rw,60 ) [Location] []
  LAT = +40 -90 +90 +0 %10.6m [Lat  D:M:S +N]
  LONG = -90 -180 +180 +0 %12.8m [Long D:M:S +E]

Property GOTOedb (Idle,rw,36 ) [Target] []
  edb = <empty> [edb format line]


=======================================================================
Device OTA

Property Focuser (Ok,rw,60 ) [Focuser] []
  Focus = +100 -500 +500 +10 %5.2f [Focus,  m]

Property Filter (Ok,rw,60 ) OneOfMany [Filter] []
  B =  Off [Blue]
  V =  Off [Visible]
  R =  On [Red]
  I =  Off [IR]
  H =  Off [HII]
  C =  Off [Clear]


=======================================================================
Device CCDCam

Property ExpValues (Idle,rw,60 ) [Exposure] []
  ExpTime = +10 +0 +600 +5 %.1f [Duration, secs]

Property TempNow (Ok,ro,60 ) [TempNow] []
  Temp = +0 +0 +0 +0 %.1f [Temp now, C]

Property SetTemp (Idle,wo,60 ) [SetTemp] []
  Target = +0 +0 +0 +0 %.1f [Target temp, C]

Property Binning (Idle,rw,60 ) OneOfMany [Binning] []
  1:1 =  On [1:1]
  2:1 =  Off [2:1]
  3:1 =  Off [3:1]
  4:1 =  Off [4:1]

Property Pixels (Idle,ro,60 ) [Images] []
  Img = <empty> [Science frame]

Property Code (Idle,wo,60 ) [Code] []
```

Rationale

If your're already familiar with the INDI specification, you should recoignize all the properties and its attributes like name, group, label, timeout, etc.

Advanced: Connection to several INDI servers

It is possible to connect to several INDI servers simultaneously by creating the necessary objects with `IndiServer:` NTOScript will handle all incoming messages and create a database of properties for each server.

Regarding output messages, it is not necessary to explicit which remote server the messages are sent to. NTOScript handles the notion of `TheCurrentServer`. It is set to the last `indiserver` object you issued a `connect`.

When maintanining several connections to remote servers, you should manually switch between one server and another when sending outgoing messages.

```
IndiServer: indiserver1 192.168.0.2 7624
indiserver1 connect

IndiServer: indiserver2 192.168.0.3 7624
indiserver2 connect

indiserver1 to TheCurrentServer
indiserver1 properties get
2 seconds waiting

indiserver2 to TheCurrentServer
indiserver2 properties get
2 seconds waiting
```

3.3.1.2 Handling properties

Example 4

Once we have connected to a remote '`indiserver`' and performed the device discovery, its time to control our devices. We will start by telling the telescope mount our geographical position:

```
 40:25:00 LAT.   property item   Mount GEOGRAPHIC_COORD LAT  set
-03:42:00 LONG.  property item   Mount GEOGRAPHIC_COORD LONG set
                 property vector Mount GEOGRAPHIC_COORD send&wait
```

In the [Example 4], page 11, we are setting the individual property items `LAT` and `LONG` that belong to property vector named `GEOGRAPHIC_COORD` in device named `Mount`.

Rationale: Properties

As we have said in Chapter 3 [INDI in nutshell], page 7, the way to control an INDI device is to get and set values for *property vectors*. Since a given *property vector* is composed of one or more *property items*, we set values item by item. We **must** know beforehand:

1. The *device* name.
2. The *property vector* name.
3. The *property item* name.

4. The *property vector* type, which determines which value types to read and write in the property items.
5. The *property item* physical units in case of *number vectors*.

Setting individual values is not enough, we must send the whole vector to the server for our action to be effective.

In [Example 4], page 11 above, we can see the basic pattern:

- `property item` scans three words after it: a device name, a property vector name and a property item name. Names are case sensitive. It returns an `INDI-ITEM` object.
- `property vector` scans two words after it: the device name and property vector name. It returns a `PROPERTY-VECTOR` object.
- words `set` and `send&wait` are methods applied to the returned object on the stack.
- parameters for the methods are placed at the beginning of the phrase.

Synchronous execution

The INDI protocol is asynchronous. There are no request/reply pattern *per se*. This is fine for GUIs. However, scripting is easier if we can make sure that we proceed one step at a time, i.e., turning on the mount and set the coodinates once we know that the mount has been turned on. And this is where the property state plays a role.

In NTOScript, there are different posibilities:

- Use `send` to send the message right away and continue with script execution. Fine for interactive use, typing commands by hand.
- Use `send&wait` to send the message and wait for a non-busy answer (`Ok`, `Idle` or `Alert`). After a timeout value has expired (property dependant), if the property state is still `Busy`, then throw an exception.
- Use `send&ok` to send the message and wait for an `Ok` answer. After a timeout value has expired (property dependant), if the property state is not `Ok`, then throw an exception.
- Use `send&idle` to send the message and wait for an `Idle` answer. After a timeout value has expired (property dependant), if the property state is not `Idle`, then throw an exception. This is useful when switching off devices, as we shall see in [Example 5], page 12.

Examples 5 and 6

Now, its time to power on the mount. This is necessary for the mount to start tracking at the current position.

```
True property item   Mount Power On set
     property vector Mount Power send&ok
```

We will also set the target coordinates (M31 galaxy). When set, the telescope will start slewing and will take a while to do so.

```
   00:42:44 R.A.  property item   Mount EQUATORIAL_COORD RA  set
  +41:16:08 DEC.  property item   Mount EQUATORIAL_COORD DEC set
                  property vector Mount EQUATORIAL_COORD send&ok
```

In the sample drivers, sending an new EQUATORIAL_COORD property will abort slewing if the property was `Busy`, but this is device dependant.

Rationale: Number Property vectors

When it comes to scripting, *Number property vectors* deserve an special note. The INDI specification is ambiguous about what data type to use for numbers (integers or floating point? what ranges?). **We have chosen the '64-bit "double floating point" data type for Number property vectors** because:

- It can represent accurately integers.
- It can obviously represent real numbers.[2].
- It has the widest dynamic range for both and enough precision (16 digits mantissa).

This has its consequences:

1. Number literals used for number propety vectors in scripts **must** have a dot `.`, even for integers values.
2. You can use integers if you convert them to floating point before assigning to a property item. The best way to do it is using or writting a custom unit. See Chapter 3 [Adding readability], page 7

Advanced: Property Timeout

As per the INDI specification, there is a timeout attribute attached to each property. It is used in `send&wait`, `send&ok` and `send&idle`. NTOScript provides a default timeout in (integer) seconds when this attribute is not sent by the remote device by using `Property-timeout`. You can customize this value to your taste. However a value set too low may raise "property in Busy state" exceptions later on.

```
Property-timeout .
10 to Property-timeout          \ 10 seconds as new default property timeout
```

3.3.1.3 Improving our skills

Example 7

Once we are on the target celestial object, its time to fine focus the camera. We will command relative movements to the focuser in a number of steps as suggested by the INDI property item that controls the focuser.

```
DEFINITION: +Focuser  ( -- )
      property item   OTA Focuser Focus get
      property item   OTA Focuser Focus step
   f+ property item   OTA Focuser Focus set
      property vector OTA Focuser send&ok
END-DEFINITION


DEFINITION: -Focuser  ( -- )
      property item   OTA Focuser Focus get
      property item   OTA Focuser Focus step
   f- property item   OTA Focuser Focus set
      property vector OTA Focuser send&ok
END-DEFINITION
```

Then, type:

[2] although not always accurately, as you probably know

```
+Focuser  +Focuser  +Focuser  -Focuser
```

The `step` method returns the sugested stepping value. Method `get` gets the current value as returned by the device. Words `DEFINITION:` and `END-DEFINITION` are explained below.

The last line shows our focusing attempt by moving the focuser three times forward and one time backwards.

Rationale: Adding readability

FORTH is traditionaly known as a "write-only" language. Admittedly, some of the FORTH words' names like `:` `;` `@` `!` `2@` `2!` `c@` or `c!` look rather intimidating at first. However, this write-only reputaion has to do more with ancient practices than with the language itself. See [Stephen Pelc], page 1 for a discussion on this issue.

I have tried to add substantial syntactic sugar for readability purposes, as a way to attract newcomers to NTOScript.

If you are familiar to FORTH, you should have no problem to reconize `definition:`, `end-definition`, `fetch`, `store`, `2fetch`, `2store`, `cfetch` and `cstore` as being synonyms for words above.

Also, specifying physical units for *Number vectors* is a good thing towards readability. Please, have a look at Chapter 6 [Other Physical/Non-physical Units], page 51. If you need something else, you can define it yourself.

Specifying angles

As seen in [Example 6], page 12, we have a convenience notation to specify sexagesimal quantities like angles using integer degrees, minutes and seconds (or arcseconds) by using `DD:MM:SS` plus a conversion word. In case of angles, it can be `R.A.`, `DEC.`, `LONG.`, `LAT.`, `AZ.` or `ALT.`

Other possible choices for specifying angles (and time) are `DD:MM`, `DD:MM.M` and `DD:MM:SS.S`. See Chapter 8 [Sexagesimal Numbers], page 69 for more details.

Specifyng Time seconds

In the code snippet below, word `sec.` is syntactic sugar and does nothing. So no integer literals are allowed, you must include a dot.

```
23. sec. property item CCDCam ExpValues ExpTime set
```

Specifying Lengths

The code snippet below case demonstrate that you should know both the phyiscal units handled by the device and what conversion is being performed by `um.`. Otheriwse, a disaster is assured.

The glossary says that `um.` converts from floating point microns to meters, but the device label shows that it wants numbers directly in microns.

Other units available are `cm.` and `mm.` which perform similar conversions.

```
150. um. property item Focuser Focus set  error
```

Specifying Integer units

In the code snippet below, `images` converts integer `25` to floating point `25.0` when setting the number of images to take for the CCD device named `AUDINE1`.

```
25 images property item AUDINE1 EXP_LIMITS COUNT set
```

Specifying Date and Time

This code snippet shows how to specify a timestamp for busy waiting until November, 11th, 2008 at 15:23:20 UTC time. Again, time is a sexagesimal quantity to be written as `HH:MM:SS`, seconds included. Time specifier `GMT` or `LT` **must** be present. dot. Use full month names. Date order is shown in the example.

```
2008 November 11  15:23:30 GMT at-time
```

Word `at-time` blocks the calling task until the specified date and time arrives.

Specifying delays

Phrases below blocks the calling task in a busy wait either in milliseconds, seconds or minutes.

```
100 ms
 34 seconds waiting
  5 minutes waiting
```

Rationale: Definitions

If you find yourself repeating over and over again the same set of words, it is more convenient to factor them into a definition,[3] that is compiled into machine code. Once you know your INDI devices, you will probably be writting reusable, parametrized definitions that you'll use over and over again. You **must** feel comfortable with stack handling to write non-trivial definitions.

[Example 7], page 13 creates two words called `+Focuser` and `-Focuser` that do not consume nor produce any number on the data or floating point stacks, and whose purpose is to advance backwards and forwards the focuser in steps described by the *property item*. The definitions above use the floating point stack to place two floating point numbers and perform an addition or substraction using `f+` or `f-`

You can use most of the words covered in this cookbook inside a definition, with the notable exception of `IndiServer:`. This word is used to define a word (`indiserver`) and you cannot use it while trying to define another word (`myconnect`) at the same time.

```
DEFINITION: myconnect ( -- )
  IndiServer: indiserver localhost 7624  error
  indiserver connect
  indiserver properties get
  2 seconds waiting
END-DEFINITION
```

Other FORTH *defining words* words that you may encounter such as `Create Constant` or `Variable`. They are use as intended *outside* a definition. They can be used inside definitions, but beware: the effect is not what you now expect, for sure !

Once you feel comfortable with FORTH you can expand NTOScript with your own definitions of any kind with FORTH's unique capabilities. From now on, most of our examples will be definitions.

We encourage you to write small definition that perform simple and well defined tasks such as `+Focuser` and `-Focuser` shown above. Also, beware of excessive stack gymnastics.

Example 8

We are almost over. Just before taking a photo, we must select the proper filter. We will command the filter wheel to setup the filter of our choice.

[3] NTOScript definitions are simply FORTH colon definitions.

```
0 Constant Blue
1 Constant Visible
2 Constant Red
3 Constant IR
4 Constant HII
5 Constant Clear

DEFINITION: Filter  ( n1 -- )
   case
     Blue    of True property item OTA Filter B set endof
     Visible of True property item OTA Filter V set endof
     Red     of True property item OTA Filter B set endof
     IR      of True property item OTA Filter I set endof
     HII     of True property item OTA Filter H set endof
                True property item OTA Filter C set
   endcase
   property vector OTA Filter send&ok
END-DEFINITION
```

Then, type

```
Visible Filter
```

to move the filter wheel and use the V filter.

Rationale: Designing phrases

Much of the FORTH abilities to counteract its reputation of "write-only code" lays in the hand of the programmer to find well thought phrases that reads well in English. This is an art and is not always possible for a variety of reasons. Can you find the right words and phrase ordering ? Are some words being already used ?

We have been lucky in the example above. We can type `Visible Filter` to command the filter wheel and set the Visible filter. Isn't that wonderful ?

Rationale: Control structures

Another reason to use definitions is to incorporate control structures like:

- `IF ... ELSE .. ENDIF`
- `DO ... LOOP`
- `CASE ... ENDCASE`
- `BEGIN ... AGAIN`
- `BEGIN ... UNTIL`
- `BEGIN ... WHILE ... REPEAT`

You **must** be comfortable with stack handling. See Chapter 1 [Reference documents], page 1 for more details about control structures.

Example 9

Our final task to take a single CCD images will be to prepare the camera for the photo. We will select a high resolution, 1x1 binning mode and will set the CCD exposure to 20 seconds. This `CCDCam` driver behaviour is to start an exposure whenever a new value is written on the `ExpValues` property vector. We must also enable BLOBs to receive the image, as they are not enabled by default.

```
DEFINITION: Photo ( -- )
  device CCDCam BLOBs enable
  True     property item   CCDCam Binning 1:1 set
           property vector CCDCam Binning send&ok
  20. sec. property item   CCDCam ExpValues ExpTime set
           property vector CCDCam ExpValues send&ok
END-DEFINITION
```

Then, type

```
Photo
```

to take your first photo.

Rationale: Logging

While waiting for the exposure to finish, the following informative messages are shown at the console:

```
2008-04-01T09:40:49: CCDCam: ExpValues: Starting 2 sec exposure binned 1:1
2008-04-01T09:40:51: CCDCam: ExpValues: Exposure complete, sending image
2008-04-01T09:40:51: CCDCam: Received file: CCD1_2008-04-01T11:40:51.fts.z
2008-04-01T09:40:51: CCDCam: uncompressed : CCD1_2008-04-01T11:40:51.fts
2008-04-01T09:40:51: CCDCam: Stored in:
2008-04-01T09:40:51: CCDCam: ExpValues: Image sent
```

The usual format of these messages is:

- an UTC timestamp,
- the device name,
- the property vector name and
- the free text message.

This can be used to redirect the script execution to a file. However, they can be turned off/on by issuing:

```
indi logging off
indi logging on
```

Advanced: INDI messages debugging

You can debug incoming INDI messages arriving at a given server. Tracing can be enabled or disabled per message class. The following incoming message types can be printed on standard output:

- defTextVector
- defNumberVector
- defSwitchVector
- defLightVector
- defBLOBVector
- setTextVector
- setNumberVector
- setSwitchVector
- setLightVector
- setBLOBVector
- indi-message

Make sure that INDI logging is on and invoke the following method on the indiserver object.

```
indi logging on
defTextVector indiserver +trace        \ enables  tracing
defTextVector indiserver -trace        \ disables tracing
```

You can also review outgoing messages issued by NTOScript. After having connected and discovered all your devices, disconnect from the remote server. At that point, outgoing messages are printed to standard output instead of being routed to the remote server. Example:

```
IndiServer: indiserver localhost 7624
indiserver connect
indiserver properties get
2 seconds waiting
property vector CCDCam Exposure send      \ instead of send&wait and friends !
```

3.3.1.4 Advanced Programming

We have concluded the quick, easy route to take a first photo. Of course, things can be more complex. Next examples introduce some advanced topics.

Example 10

Our final example will use the Weather station to monitor and react upon adverse conditions. Each time the wind or humidity exceeds one threshold the property `WX` in device `Weather` enters into `Alert` state. We will monitor this property and turn on and off the telescope power. Since we have no event driven programming inNTOScript, we have to design a separate thread or “task” that periodically polls this property.

```
True  Constant switch-on
False Constant switch-off

DEFINITION: Telescope-on ( -- )
   s" switching on telescope mount" background logtask
   True property item   Mount Power On set
        property vector Mount Power send&ok
END-DEFINITION

DEFINITION: Telescope-off ( -- )
   s" switching off telescope mount" background logtask
  True property item   Mount Power Off set
       property vector Mount Power send&idle
END-DEFINITION

DEFINITION: Telescope ( flag -- )
  IF
    property item Mount Power On get
    0= IF Telescope-on endif
  ELSE
    property item Mount Power Off get
    0= IF Telescope-off ENDIF
  ENDIF
END-DEFINITION

DEFINITION: (WeatherAction) ( -- )
  default-io decimal
  background logging on
  s" started" background logtask
  BEGIN
     2 seconds waiting
     property vector Weather WX state
     CASE
        Alert OF switch-off telescope ENDOF
        Ok    OF switch-on  telescope ENDOF
     ENDCASE
  AGAIN
END-DEFINITION

DEFINITION: WeatherAction ( -- )
  Assign (WeatherAction) catch
  background .#excp
  s" finished" background logtask
END-DEFINITION
```

Then, type

```
switch-on telescope
Task Weather-Task
Assign WeatherAction To-Start Weather-Task
```

to power on the mount (if it wasn't already) and start the weather task.

Rationale: Factor, factor, factor

The more sophisticated actions you want your devices to do, the more important is to break the problem in small pieces. Do not let definitions to take more than a few lines. Do not nest control structures in the same definition, instead make a new definition out of them. With a little practice, you will notice that each new definition can be reused in a different context. **Learn to use the stack**. It is key to write small, reusable definitions.

Words `Telescope-on` and `Telescope-off` do precise things: to turn on or off the mount. Word `Telescope` that checks if the mount was already in the desired state, to avoid redundant commands. This later takes a flag, either `switch-on` or `switch-off` that tell us what is our intention.

We also have words `(WeatherAction)` and `WeatherAction`. The former is where the real action code for the task is written. The latter wraps the action into exception handling code.

Other utility words being used: `default-io`, `decimal`, `background`, `logtask`. See Chapter 6 [Glossary], page 51 or *VFX Forth for Linux User Manual* for their descriptions.

Rationale: Designing tasks

Multitasking can be a bit intringing. Tasks can be as straightforward or as complex as you can imagine. Debugging tasks can be tricky. Your task should have at least a top level exception handler. Otherwise, your task may die and you may even not notice. Although you can write tasks that start and finish cleanly, tasks are usually endless loops. In this case, it is of utmost importance to watch out stack imbalances.

Some tools for debugging tasks are the `.s`, `f.s` words that print the main stack and floating point stack contents.

You also can use the `background logtask` phrase to display task messages. “background messages” are a kind of messages available to the programmer as opossed to “indi messages” which display the INDI activity. Background messages must be turned on (globally, for all tasks).

```
background logging on
s" Message from task" background logtask
```

will display

```
2008-04-01T09:40:51 Task <taskname> Message from task
```

Introducing tasks may introduce complexity: one task may depend on the activity of another task. This must be taken into account and there are several mechanisms to synchronize tasks. [Example 11], page 20 shows one example. Another issue to take care is to define `USER` variables (when necessary) instead of ordinary variables.

See [Stephen Pelc], page 1 and [VFX Forth for Linux User Manual], page 1 for a detailed discussion on multitasking.

Example 11

If you are still with us, let us introduce to you a better refactored example:

- We will introduce an autofocuser task. The autofocuser task simulates some actions of the HFD autofocusing algorithm by Steve Brady and Larry Weber. This task will start each 90 seconds, if an only if the CCD is not being used for ordinary imaging a the moment.
- We will refactor the image adquisition code to reuse it from two tasks.

As this example is rather long, it is splitted in several “cartouches”.

```
DEFINITION: discover   ( -- )
  indiserver connect
  indiserver properties get
  2 seconds waiting
END-DEFINITION

\ -----------------
\ *H Mount Handling
\ -----------------

DEFINITION: My-Location  ( -- )
   40:25:00 LAT.   property item   Mount GEOGRAPHIC_COORD LAT  set
  -03:42:00 LONG.  property item   Mount GEOGRAPHIC_COORD LONG set
                   property vector Mount GEOGRAPHIC_COORD send&idle
END-DEFINITION

DEFINITION: Goto-M31  ( -- )
   00:42:44 R.A.  property item   Mount EQUATORIAL_COORD RA  set
  +41:16:08 DEC.  property item   Mount EQUATORIAL_COORD DEC set
                  property vector Mount EQUATORIAL_COORD send&ok
END-DEFINITION

DEFINITION: Telescope-on ( -- )
   s" switching on telescope mount" background logtask
   True property item   Mount Power On set
       property vector Mount Power send&ok
END-DEFINITION

DEFINITION: Telescope-off ( -- )
   s" switching off telescope mount" background logtask
  True property item   Mount Power Off set
       property vector Mount Power send&idle
END-DEFINITION

True  CONSTANT switch-on     \ flags for the word below
False CONSTANT switch-off

DEFINITION: Telescope ( flag -- )
  IF
    property item Mount Power On get
    0= IF Telescope-on ENDIF
  ELSE
    property item Mount Power Off get
    0= IF Telescope-off ENDIF
  ENDIF
END-DEFINITION
```

We have conveniently refactored the conection and discovery steps into word `discover` that do not take any parameters.

```
\ ------------------
\ *H Camera Handling
\ ------------------

semaphore ccdsem
ccdsem    InitSem

1 CONSTANT 1x1
2 CONSTANT 2x2
3 CONSTANT 3x3
4 CONSTANT 4x4

DEFINITION: Photo ( binning -- ; F: exptime -- )
  CASE
    1x1 OF True property item CCDCam Binning 1:1 set ENDOF
    2x2 OF True property item CCDCam Binning 2:1 set ENDOF
    3x3 OF True property item CCDCam Binning 3:1 set ENDOF
    4x4 OF True property item CCDCam Binning 4:1 set ENDOF
  END-CASE
  property vector CCDCam Binning send&ok
  property item   CCDCam ExpValues ExpTime set
  property vector CCDCam ExpValues send&ok
END-DEFINITION

DEFINITION: Photos ( binning nphotos --  F: exptime -- )
  ccdsem request
  device CCDCam BLOBs enable
  auto-blobs-dir property vector CCDCam Pixels directory set
  0 DO
    dup fdup Photo
  LOOP
  drop fdrop
  ccdsem signal
END-DEFINITION
```

We have generalized `Photo` to take the binning mode and exposure time as parameters (on the data and floating point stack respectively).

Binning mode words are defined like constants. Unlike other languages, FORTH allows names with any character not being a <SPACE>, <TAB> or <CR>

A difference with the previous version of `Photo` is that "enable BLOBs" message has been taken out. There is no need to send it every time.

We would like to have a word that takes a series of photos for two purposes:

- for your imaging session and
- for the autofocuser task

Of course, this word is named `Photos` and simply loops over `Photo`, taking one parameter more, the number of photos to take. It is customary to place the argument being first used on the top of the stack.

Incoming BLOBs such as CCD images are left in the current directory where NTOScript is launched, unless an specific directory is set.

The phrase

`property vector CCDCam Pixels directory set` creates and sets a directory given by a string placed in the data stack.

For convenience, we have included word `auto-blobs-dir` which generates a directory string with the format:

$HOME/ccd/<julian day>

where $HOME is the user's HOME enviroment variable. This naming scheme based on julian day has the property of being invariant throughout a whole night of imaging, removing the ambiguity of where to place images before and after midnight.

You may have noticed the declaration of a `SEMAPHORE` at the beginning. As you probably know, this is a familiar inter-task synchronization mechanism. It ensures that the autofocusing task do not interfere when doing an imaging session of, let's say `2x2 60. sec. 5 Photos`.

```
\ -------------------
\ *H Focuser Handling
\ -------------------

DEFINITION: +Focuser  ( -- )
      property item   OTA Focuser Focus get
      property item   OTA Focuser Focus step
   f+ property item   OTA Focuser Focus set
      property vector OTA Focuser send&ok
END-DEFINITION

DEFINITION: -Focuser  ( -- )
      property item OTA Focuser Focus get
      property item OTA Focuser Focus step
   f- property item OTA Focuser Focus set
      property vector OTA Focuser send&ok
END-DEFINITION

0 CONSTANT Blue
1 CONSTANT Visible
2 CONSTANT Red
3 CONSTANT IR
4 CONSTANT HII
5 CONSTANT Clear

DEFINITION: Filter  ( n1 -- )
   CASE
     Blue    OF True property item OTA Filter B set ENDOF
     Visible OF True property item OTA Filter V set ENDOF
     Red     OF True property item OTA Filter R set ENDOF
     IR      OF True property item OTA Filter I set ENDOF
     HII     OF True property item OTA Filter H set ENDOF
        True property item OTA Filter C set
   ENDCASE
   property vector OTA Filter send&ok
END-DEFINITION
```

Basic Focuser and Filter wheel handling code is the same as before ([Example 8], page 15. However, next example will introduce our simulated autofocusing process.

```
\ -----------------
\ *H Autofocus Task
\ -----------------

DEFINITION: VShape ( -- )
  16 0 DO
     2700 ms
     +Focuser
      s" taking a hi-res focus image" background logtask
     5.0 sec. 1x1 Photo
  LOOP
END-DEFINITION

DEFINITION: BestPosition ( -- )
    8 0 DO 2300 ms -Focuser LOOP
END-DEFINITION

DEFINITION: AutoFocus ( -- )
    ccdsem request
    VShape BestPosition
    ccdsem signal
END-DEFINITION

DEFINITION: (Autofocuser) ( -- )
  default-io decimal
  background logging on
  s" started" background logtask
  BEGIN
     90 seconds waiting
     AutoFocus
  AGAIN
END-DEFINITION

DEFINITION: Autofocuser ( -- )
  Assign (Autofocuser) CATCH
  ?dup IF
     background .#excp ccdsem initSem
  ENDIF
  s" finished" background logtask
END-DEFINITION
```

This series of definitions simulate a brute force Half Flux Method autofocusing algorithm:

- The calibration photo series to obtain an V "HFD versus position" shape is defined in `VShape`
- The final move to the best position just found is defined in
- An autofocusing run is defined in `AutoFocus`.
- The real autofocuser task that initializes task variables and runs every 90 seconds is defined in `(AutoFocuser)`
- The wrapper task action that handles exception codes thrown is `Autofocuser BestPosition`.

As mentioned before, each autofocus run must be done if and only if no imaging session is in progress, so it uses the `ccdsem` semaphore.

```
\ ---------------
\ *H Weather Task
\ ---------------

DEFINITION: (WeatherAction) ( -- )
  default-io decimal
  background logging on
  s" started" background logtask
  BEGIN
     2 seconds waiting
     property vector Weather WX state
     CASE
        Alert OF switch-off telescope ENDOF
        Ok    OF switch-on  telescope ENDOF
     ENDCASE
  AGAIN
END-DEFINITION

DEFINITION: WeatherAction ( -- )
  Assign (WeatherAction) CATCH
  background .#excp
  s" finished" background logtask
END-DEFINITION
```

The weather task is the same as before.

Rationale: Including files

Your final imaging script can be quite concise and it is composed of words, tailored to your INDI drivers, performing a good amount of work and that you can reuse over and over again. You can place them in a separate file, lets say '`xephem.fs`' and simply include it.

```
include xephem.fs
```

```
\ --------------
\ *H Main Script
\ --------------

foreground logging on
IndiServer: indiserver localhost 7624
include xephem.fs     \ file where the above definitions live
discover

Task Weather-Task
Assign WeatherAction To-Start Weather-Task

Task AutoFocus-Task
Assign Autofocuser To-Start AutoFocus-Task

switch-on Telescope
My-Location
Goto-M31

Red Filter
30.0 sec. 2x2 3 Photos

Visible Filter
30.0 sec. 2x2 3 Photos

Blue Filter
60.0 sec. 2x2 3 Photos

Clear Filter
60.0 sec. 1x1 3 Photos
```

3.3.2 Sky Mapping package

The sky mapping package provides a tool to generate (R.A., Dec.) coordinate pairs that sample "rectangular" areas in the sky for various tasks like mosaics or asteroid hunting. It includes the necessary declination correction to account for the noticeable decreasing RA arc length produced at medium to high declinations. (see Figure 3.1 and Figure 3.2).

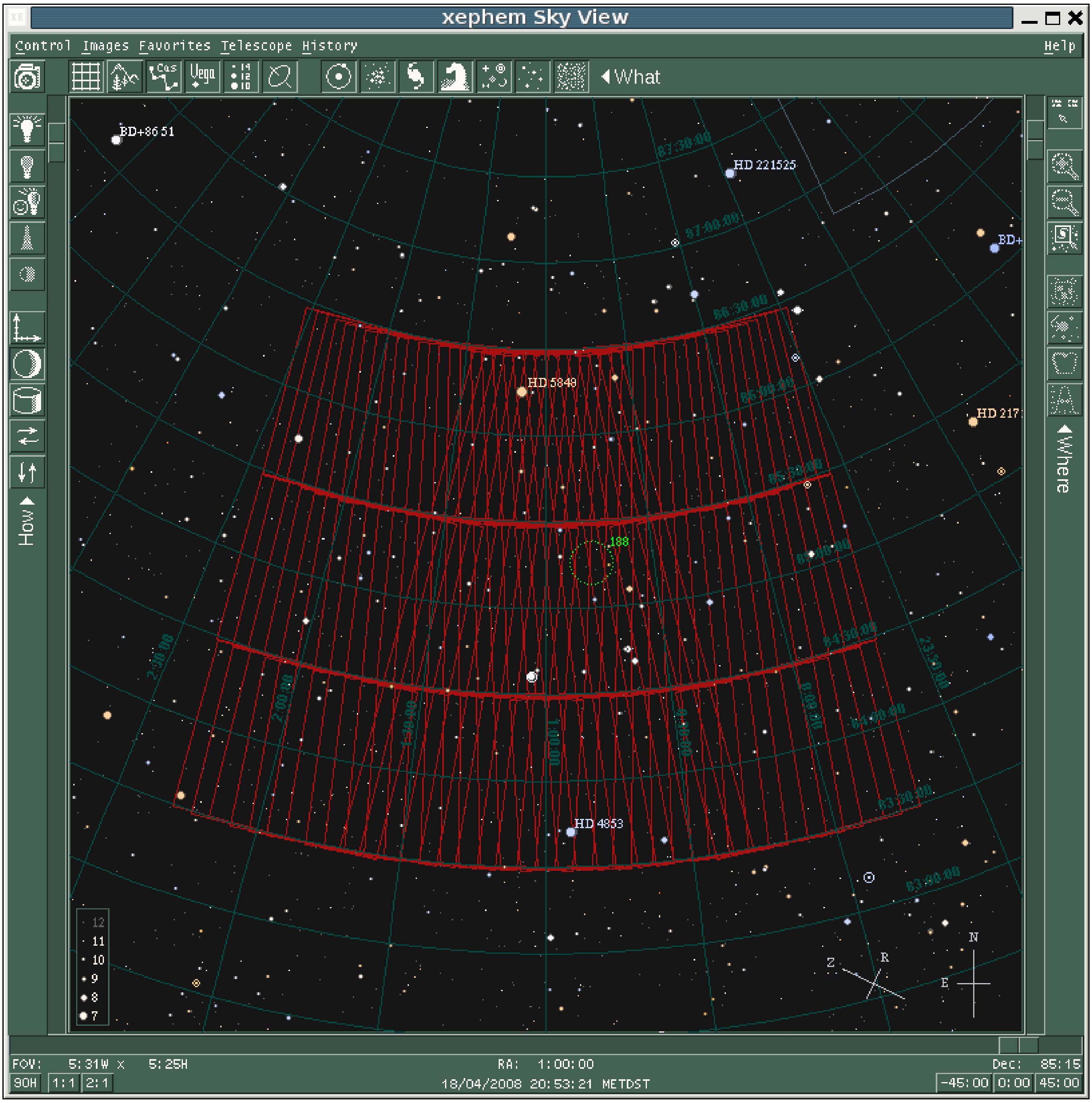

Figure 3.1: Sky frames around NGC 188 produced without declination correction.

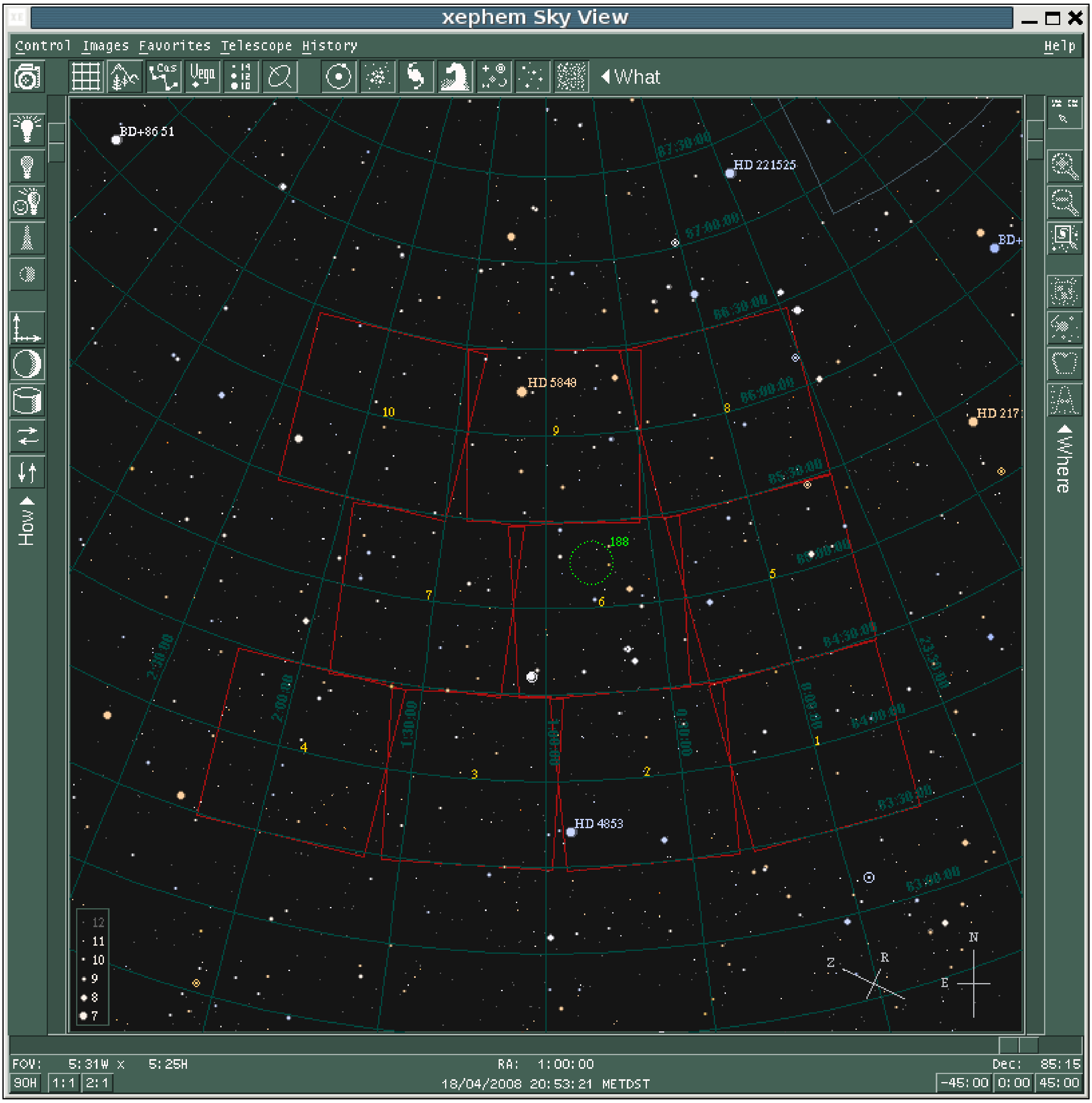

Figure 3.2: Sky frames around NGC 188 produced with declination correction. In fact, there are still gaps by not applying the correction respect to the frame edge.

Coordinates generation (sampling) can be according to the following patterns:

- `UNIFORM` pattern
- `ZIGZAG` pattern

```
>---------*      +---------*
                 |
>----x---->      +----x----+
                           |
o--------->      o---------+
```

Figure 3.3: Sky Mapper Uniform and ZigZag patterns.

and each pattern in different modes, which account for the differenct combinations: `+RA +DEC`, `-RA -DEC`, `+RA -DEC` and `-RA +DEC`. The fast moving axis is always RA.

For a given set of initial and final (R.A.,Dec.) coordinates, there are two possibilities for mosaicing the zone. To resolve this ambiguity we have introduced a limitation. The mosaicing zone should not cover more than 12:00:00 hours in R.A. The mapping algorithm always choose the smaller zone.

3.3.2.1 Planning the mosaic

We will continue working with the example given in the [Example 11], page 20. Now the task is to do an LRGB mosaic of M31 galaxy through `Clear`, `Red`, `Visible` and `Blue` filters. Unlike the previous example, we will not use any independent task for focusing. Instead we will reuse the `Autofocuser` action to explicitely focus after each filter movement.

The very first thing to do is to study wether this is feasible given the CCD chip area and telescope focal length. NTOSCRIPT includes a couple of utility words, `CCDChip:` and `OTA:` to define data structures that help determining the angular size seen from the chip.

```
4904 x 3280  7.4 um. x 7.4 um. CCDChip: SXVF-H36
80.0 inch. x 8.0 inch.         OTA:     LX200-8"
```

As seen from the example, the `CCDChip:` word defines a data structure named `SXVF-H36` and needs the horizontal and vertical resolutions[4] and the dimensions of each physical pixel. The `OTA:` word defines a data structure named `LX200-8"` and needs the focal length and diameter (in this order). The internal data structures work in international standard units (meters), so is necessary to perform conversions. This is easy using unit words like `um.` or `inch.`

Examining the zone with our favourite planetarium, we choose as a suitable big frame the sky area covered between `00:50 R.A. 43:00 DEC.` and `00:35 R.A. 39:30 DEC.` . We will sweep the sky by decreasing R.A. cordinates and increasing Dec. coordinates. Now, we are ready to define our skymapper object. It is also convenient to specify a slight amount of overlap in both axes to align each frames when composing the mosaic.

```
-RA +DEC Uniform SkyMapper: skymapper1
00:50 R.A.    43:00 DEC.      skymapper1 set-initial
00:35 R.A.    39:30 DEC.      skymapper1 set-final
LX200-8"      SXVF-H36   FOV  skymapper1 set-frame
   10 %RA     10 %DEC         skymapper1 set-overlap
```

When setting the individual frame size with `set-frame`, the R.A. and Dec. angular sizes must be expressed in degrees. This is automatically supplied by the word `FOV` which takes a CCD chip structure and an OTA structure in this order. Overlap is specified as an integer from 0 to 99.

[4] word `x` is just syntactic sugar

The other choice to initialize a sky mapper is specifying a field centre point and the number of frames to take in each axis. See Chapter 7 [Creating sky mapper objects], page 59

It is possible to graphically depict what is the mapped area and find out by yourself if this is feasible. Defining word `XEphem-Tool:` creates tool instances. A given tool instance generates two files:

- A eyepieces location file named '`mapping.epl`' and
- An annotations file named '`mapping.ano`', with the sequence numbers

These two files are placed under the user's '`$HOME/.xephem`' directory.

To do this, just type:

```
XEphem-Tool: xephem
skymapper1 xephem set-mapper
xephem generate
```

Pay attention to the foreground logging, as it tells the number of frames generated to map the desired sky area.

```
2008-04-20T23:06:46: 24 samples for eyepieces & annotations.
```

Now, launch XEphem and select your favourite SkyView from the History menu. Then display the mapped area. You can do this either by manually scrolling or using XEphem data index facility or using again the NTOScript tool. See Chapter 7 [XEphem tool], page 59 for more details.

For the example above, we have the following mapped area:

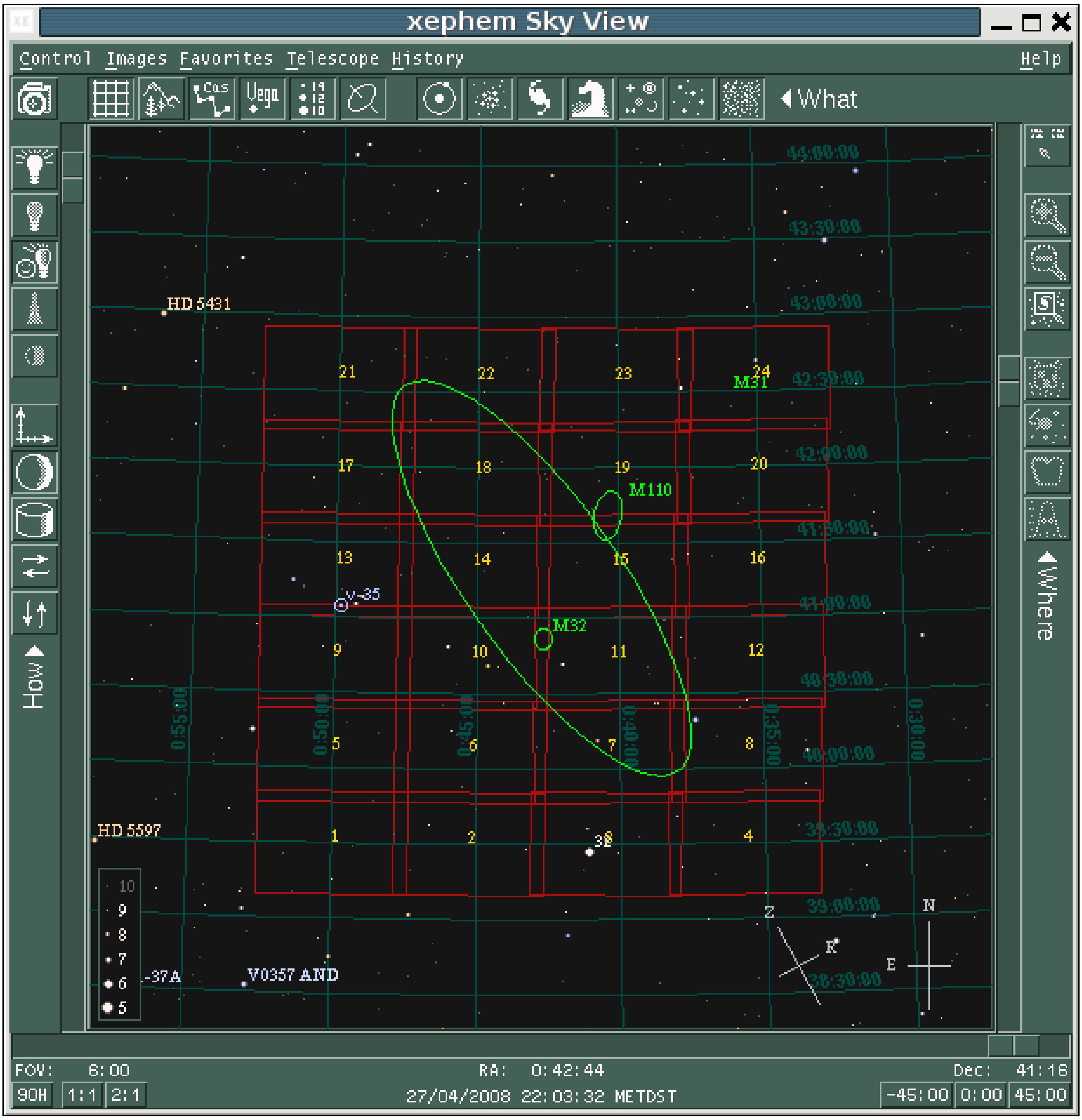

Figure 3.4: Planned mosaic of M31 field using an LX200 8" OTA and an Starlight SXVF-H36 CCD

3.3.2.2 Sweeping the sky

Once we have created our sky mapper object and verified the feasibility of our mosaic, we can proceed to program the script for our mosaic, but first a couple of handy definitions.

```
DEFINITION: GoTo-RADEC  ( F: dec ra -- )
  property item   Mount EQUATORIAL_COORD RA  set
  property item   Mount EQUATORIAL_COORD DEC set
  property vector Mount EQUATORIAL_COORD send&ok
END-DEFINITION
```

This definition will take any floating point R.A. and Dec. coordinates and send them to the mount, causing it to slew. Note that parameters are passed in the reversed order with respect to the customary (R.A.,Dec.) sequence.

```
DEFINITION: Home-Location  ( geopoint -- )
   GeoPoint@
   property item   Mount GEOGRAPHIC_COORD LAT  set
   property item   Mount GEOGRAPHIC_COORD LONG set
   property vector Mount GEOGRAPHIC_COORD send&idle
END-DEFINITION
```

This definition sends the mount the coordinates of a geographical point and avoids hardcoding coordinates inside the definition. Geographical points are defined with `GeoPoint:` see Chapter 7 [/GEOPOINT structure], page 59.

Next definition will manage a series of LRGB exposures. We will assume that we know from experience what is the exposure balance between filters to do a proper colour balancing. The only interesting parameter to play with is the number of exposures in each filter. To make it easy, we write a definition that uses the same number for all exposures. We also perform autofocusing explicitelly assuming that filters have uneven thickness.

```
DEFINITION: LRGB-Series  ( nphotos -- )
     >R
     Red Filter
     Autofocuser
     30.0 sec. 2x2 R@ Photos
     Green Filter
     Autofocuser
     40.0 sec. 2x2 R@ Photos
     Blue Filter
     Autofocuser
     60.0 sec. 2x2 R@ Photos
     Clear Filter
     Autofocuser
     30.0 sec. 1x1 R> Photos
END-DEFINITION
```

We have used the *return stack* to keep the number of photos, although we could have used the *locals* facility for easier handling and readability.

Now, we are ready to write our mosaic loop that uses the `skymapper1`:

```
DEFINITION: LRGB-Mosaic ( -- )
  skymapper1 reset
  BEGIN
     skymapper1 next-point
  WHILE
     EqPoint@ GoTo-RADEC
     10 seconds waiting
     5 LRGB-Series
  REPEAT
END-DEFINITION
```

Note that the very first thing one should do to a sky mapper object after setting its points is to `reset` it.

Method `next-point` in `skymapper1` returns an `/EqPoint` structure and `True` or simply `False` when this object finishes generating coordinates. Floating point coordinates in this structure are fetched in reverse order using the word `EqPoint@`. These values are fed into `Goto-RADEC` to move the `Mount` device and finally we take 5 exposures per filter. We also wait 10 seconds for the telescope to stabilize tracking.

And finally, our M31 mosaic main script is simply:

```
foreground logging on

IndiServer: indiserver localhost 7624
include xephem.fs

-03:42 LONG.  40:25 LAT. GeoPoint: Madrid

4904 x 3280  7.4 um. x 7.4 um. CCDChip: SXVF-H36

80.0 inch. x 8.0 inch. OTA: LX200-8"

-RA +DEC Uniform SkyMapper: skymapper1

00:50 R.A.   43:00 DEC.     skymapper1 set-initial
00:35 R.A.   39:30 DEC.     skymapper1 set-final

LX200-8"     SXVF-H36  FOV  skymapper1 set-frame
   10 %RA      10 %DEC      skymapper1 set-overlap

discover
switch-on telescope
Madrid Home-Location
LRGB-Mosaic
```

4 Properties database

4.1 Introduction

The heart of INDIScript is the INDI properties database. This is an in-memory, hierarchical database of objects with four levels:

1. The server level. There are as many `INDI-SERVER` objects as connections to remote INDI servers.
2. The device level. All devices under a given server are placed together as children of a given `INDI-SERVER` object. All properties belong to a given device and are placed under its parent `INDI-DEVICE`.
3. The property level. This is where all `PROPERTY-VECTOR` objects live.
4. The property item level. The individual components of each `PROPERTY-VECTOR`.

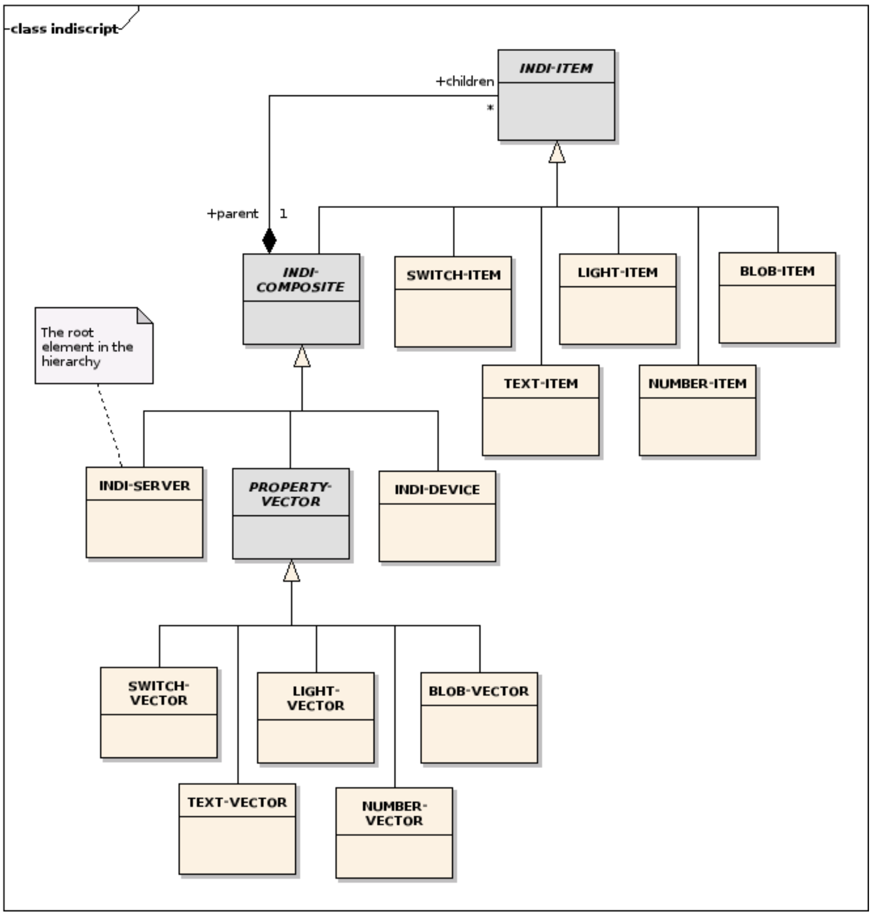

Figure 4.1: Class hierarchy.

The properties database API is placed in the `INDI` vocabulary. Details and design notes for each class are given in the following glossary section.

The object system choosen by INDIScript - *objects.fs* - has been ported from *GForth* to *VFX Forth for Linux*. Please, read the GForth manual for details about this OOP package.

4.2 Glossary

4.2.1 INDI-ITEM class

The `INDI-ITEM` class is the base class for all the properties database levels. As such, it must **not** be directly instantiated.

```
: INDI-VERSION                          \ -- ca u
```
Returns the current protocol version string supported by this client library.

Public methods

```
:m name   ( this -- ca1 u1 )
```
Get the item's name string.

```
:m parent   ( this -- obj )
```
Get the item's parent object.

```
:m label   ( this -- ca1 u1 )
```
Get the item's label string. Useful for GUIs.

```
m:  ( this -- addr )                   \ overrides next-item
```
Get the address of the next `indi-item` object for further `!` or `@`. Useful for getting siblings of a given object in GUIs but not for scripts. **Warning:** This method is not thread-safe.

```
m: ( this -- )                         \ overrides send
```
Send the most characterisict INDI message to a server. This is an abstract method with a default implementation. in which subclasses overrides `begin-xml` `body-xml` and `end-xml` with the proper contents.

4.2.2 INDI-COMPOSITE class

`INDI-COMPOSITE` class is the base class for all container classes that have other objects embedded. As such, it should not be directly instatiated. Usable descendants of this class are:

- `INDI-SERVER`.
- `INDI-DEVICE`.
- The different property vectors (`TEXT-VECTOR`, `NUMBER-VECTOR`, etc.)

Public methods

```
m:   ( ca1 u1 this -- )                \ overrides construct
```
Creates a new `indi-composite` object with name given by *ca1 u1*.

```
m:   ( this -- )                       \ overrides get
```
Get properties for this server/device/property.

```
:m size   ( this -- u )
```
Get children objects' count.

```
m:   ( this --  )                      \ overrides print
```
Recursively print children `INDI-ITEM`s.

```
m:   ( this --  )                      \ overrides latch
```
Recursively copy ive values onto hold values in children objects.

```
:m search-item   ( ca1 u1 this -- obj true | false )
```
Search for a given `INDI-ITEM` by name in its collection. The search is case sensitive. Not thread-safe version, it is intended for the `Listener` task.

```
:m lookup  ( ca1 u1 this -- obj true | false )
```

Search for a given `INDI-ITEM` by name in its collection. The search is case sensitive. Thread-safe version, uses a lock. and it is intended for the `MAIN` task.

Other helper words

```
: find-item                               \ ca1 u1 obj1 n1 -- obj2
```

Find a children object *obj2* whose name is *ca1 u1* in parent object *obj1*. Throw *n1* exception code if not found.

4.2.3 `INDI-SERVER` class

The `INDI-SERVER` class is the root class in the hierarchy of `INDI-ITEM`s. The user must first create an `INDI-SERVER` object for further interactions. The user permanently binds an `INDI-SERVER` object to a remote host and port by passing the proper parameters in the constructor. However, the connection is not made until a `CONNECT` is attempted. At this moment, the socket is open and the background `LISTENER` is started. When the user invokes the `DISCONNECT` method, the socket is closed and the background task is `HALT`ed.

When the `INDI-SERVER` object is disconnected, any `SEND` method will send the XML stream to `XConsole`. In the current implementation, any I/O error in the background `Listener` task causes this task to `STOP` and the socket is closed. No inter-task communication exist so the `MAIN` task does not notice anything until doing an I/O itself, which will throw an exception.

The following public selectors/methods may be invoked by the `MAIN` task:

- `CONSTRUCT`
- `CONNECTED?`
- `CONNECT`
- `DISCONNECT`
- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`)
- `SEND` (inherited from `INDI-ITEM`)
- `GET` (inherited from `INDI-COMPOSITE`)
- `LATCH` (inherited from `INDI-COMPOSITE`)
- `SIZE` (inherited from `INDI-COMPOSITE`)
- `PRINT` (inherited from `INDI-COMPOSITE`)
- `LOOKUP` (inherited from `INDI-COMPOSITE`)

```
NULL Value TheCurrentServer
```

Current server being handled by the scripts or other console operations.

```
ErrDef Server-Excp        "INDI Server not found"
```

Exception thrown on `NULL` value for `TheCurrentServer`.

Public methods

```
  m:   ( ca1 u1 u3 ca2 u2 this -- )           \ overrides construct
```

Create and `INDI-SERVER` object and associated background task. Object name is given by *ca2 u2*. Host is given in *ca1 u1* string. TCP port is given in *u3*.

```
  m:   ( this -- flag )                   \ overrides connected?
```

Tell connection status. True if connected.

```
m:   ( this --  )                      \ overrides connect
```
Open a socket to host/port given in the `INDI-SERVER` constructor. Also creates a background task to listen to incoming data.

```
m:   ( this --  )                      \ overrides disconnect
```
Disconnect from server. Do nothing if already disconnected.

Other helper words

```
: (INDIServer)                 \ ca1 u1 ca2 u2 ca3 u3 -- ;  [child] -- obj
```
Create a named `INDI-SERVER` object whose name is *ca1 u1* that will connect to host given by *ca2 u2* and TCP port string *ca3 u3*. Invoking the object name name will return the object handle *obj*.

```
: CurrentServer                   \ -- obj
```
Return `theCurrentServer` value or throw a `Server-Excp` exception if `NULL`.

4.2.4 `INDI-DEVICE` class

An `INDI-DEVICE` object is the second level iin the hierarchy of `INDI-ITEM`s. These objects are instantiated on demand by the background `LISTENER` task when new properties are discovered.

Two interesting usages of a given `INDI-DEVICE` are:

- To enable/disable BLOBs from this device.
- To discover properties for this device only.

The following public selectors/methods may be invoked by the `MAIN` task:

- `ENABLE`
- `DISABLE`
- `PRINT`
- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`)
- `GET` (inherited from `INDI-COMPOSITE`)
- `SEND` (inherited fro `INDI-ITEM`)
- `COPY` (inherited from `INDI-COMPOSITE`)
- `SIZE` (inherited from `INDI-COMPOSITE`)
- `LOOKUP` (inherited from `INDI-COMPOSITE`)

Public methods

```
:m enable   ( addr -- )
```
Enable BLOBs for this device. Use as follows:

```
<a device object> BLOBs enable
```

```
:m disable   ( addr -- )
```
Enable BLOBs for this device. Use as follows:

```
<a device object> BLOBs disable
```

```
m:   ( this --  )                      \ overrides print
```
Print all device properties.

Other helper words

`ErrDef Dev-Excp "INDI Device not found"`

`: (device) \ ca1 u1 -- obj`

Find an `INDI-DEVICE` whose name is given by *ca1 u1*. Throw a `Dev-Excp` exception if not found. Uses the global context given by the `CurrentServer`.

4.2.5 `PROPERTY-VECTOR` class

The `PROPERTY-VECTOR` class is another abstract class for the different property vectors encapsulating common attributes. As such, it should not be directly instantiated.

Common attributes to all property vectors:

- **name**. A string. Never modified (no need to lock).
- **label**. A string. Never modified (no need to lock).
- **group**. A string. Never modified (no need to lock).
- **permission**. An integer. Never modified (no need to lock).
- **timeout**. An integer. Can be modified (needs a lock).
- **state**. An integer. Can be modified (needs a lock).

In addition, `SWITCH-VECTOR`s have:

- **rule**. An integer. Never modified (no need to lock).

By exception, `LIGHT-VECTOR`s **do not** have:

- **permission**.
- **timeout**.

INDI Exceptions

```
ErrDef Ok-Excp    "Property in OK state"
ErrDef Busy-Excp  "Property in BUSY state"
ErrDef Alert-Excp "Property in ALERT state"
ErrDef Idle-Excp  "Property in IDLE state"
```

Property vector attributes and values

`60 Value PROPERTY-TIMEOUT`

The default timeout in seconds to wait for a property's state change from `BUSY` to some other value.

`0 Constant IDLE`

The IDLE or 'grey' property state.

`1 Constant OK`

The OK or 'green' property state.

`2 Constant BUSY`

The BUSY or 'yellow' property state.

`3 Constant ALERT`

The ALERT or 'red' property state.

`: >$state ( n -- ca1 u1)`

Given the state number *n*, gets its string literal.

```
: >state  ( ca u - n True | false )
```
Translates `"Idle"`, `"Ok"`, `"Busy"`, `"Alert"` strings to integer constants used for lights and property states. *flag* is true upon succesful conversion.

```
: >$permission   ( n -- ca1 u1)
```
Given the permission constant *n*, `r/o`, `w/o` and `r/w`, gets its string literal.

```
: >permission  ( ca u -- n True | False )
```
Translates `"ro"`, `"wo"` and `"rw"` strings to integer constants `r/o`, `w/o` and `r/w`. Returned flag is true upon succesful conversion.

Public methods

```
m:   ( ca1 u1 this -- )                      \  overrides construct
```
Create a new `PROPERTY-VECTOR` object with name given by *ca1 u1*.

```
:m state   ( this -- n1 )
```
Get the property vector's state. To be used by the `MAIN` task.

```
m:   ( this -- n1 )                          \ overrides timeout
```
Get the property vector's timeout. To be used by the `MAIN` task.

```
m:   ( this -- n1 )                          \ overrides permission
```
Get the property vector's permission. To be used by the `MAIN` task.

```
:m group   ( this -- ca u )
```
Get the property vector's group. To be used by the `MAIN` task.

```
m:    ( this -- )                            \ overrides send
```
Send `<newXXXVector...> ... </newXXXVector>` and then mark the property vector's state as `BUSY`.

```
m:   ( this -- )                     \ overrides wait
```
Wait until property'state becomes non-busy or until the property's `timeout` have elapsed. Calls `PAUSE`.

```
:m send&wait   ( this -- )
```
Convenient method to invoke `send` and `wait` for the answer. Guarantees that the @fo{STATE} is not `BUSY` or else it throws `Busy-Excp`.

```
:m send&ok   ( this -- )
```
Convenient method to invoke `send` and `wait` for the answer. Guarantees that the property `STATE` is `OK` or else it throws `Busy-Excp`, `Idle-Excp` or `Alert-Excp`.

```
:m send&idle   ( this -- )
```
Convenient method to invoke `send` and `wait` for the answer. Guarantees that the property `STATE` is `IDLE` or else it throws `Busy-Excp`, `Ok-Excp` or `Alert-Excp`. Useful when switching devices off.

```
m: ( this --  )                      \ overrides print
```
Print to standard output the property name, state and individual property item names and values.

Other helper words

```
ErrDef Prop-Vector-Excp "Property vector not found"
```

```
: (vector)                         \ ca1 u1 ca2 u2 ---
```

Find a property vector whose name is given by *ca1 u1* in device whose name is given by *ca2 u2*. Can throw a `Dev-Excp` or `Prop-Vector-Excp` exceptions if any object is not found. Uses the global context given by the `CurrentServer`.

```
ErrDef Prop-Item-Excp   "Property item not found"
```

4.2.6 `TEXT-VECTOR` class

The `TEXT-VECTOR` class is the container for all the `TEXT-ITEM` objects contained within. It can be directly instantiated.

The following public selectors/methods may be invoked by the `MAIN` task:

- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`
- `GROUP`(inherited from `PROPERTY-VECTOR`)
- `STATE`(inherited from `PROPERTY-VECTOR`)
- `TIMEOUT`(inherited from `PROPERTY-VECTOR`)
- `GET` (inherited from `PROPERTY-VECTOR`)
- `SEND` (inherited from `PROPERTY-VECTOR`)
- `WAIT` (inherited from `PROPERTY-VECTOR`)
- `SEND&WAIT` (inherited from `PROPERTY-VECTOR`)
- `SEND&OK` (inherited from `PROPERTY-VECTOR`)
- `SEND&IDLE` (inherited from `PROPERTY-VECTOR`)
- `PRINT` (inherited from `PROPERTY-VECTOR`)
- `COPY` (inherited from `INDI-COMPOSITE`)
- `SIZE` (inherited from `INDI-COMPOSITE`)
- `LOOKUP` (inherited from `INDI-COMPOSITE`)

4.2.7 `TEXT-ITEM` class

The `TEXT-ITEM` class is the contained class of `TEXT-VECTOR` It can be directly instantiated.

The following public selectors may be invoked by the `MAIN` task:

- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`
- `SET`
- `GET`
- `COPY`
- `PRINT`
- `SEND`

Public methods

```
m:   ( ca1 u1 this -- )                \ overrides set
```
Set the new value ('hold' value) for this `TEXT-ITEM`.

```
m:   ( this -- ca1 u1 | x 0 )          \ overrides get
```

Get the new 'live' value for this `TEXT-ITEM`. If no value is yet available, returns 0 as string length.

```
m:   ( this -- )                        \ overrides print
```

Print `TEXT-ITEM` name and value to the standard output.

```
m:    ( this -- )                       \ overrides copy
```

Copy 'live' value into 'hold' value. If no 'live' value, empty the 'hold' value.

4.2.8 `NUMBER-VECTOR` class

The `NUMBER-VECTOR` class is the container for all the `NUMBER-ITEM` objects contained within. It can be directly instantiated.

The following public selectors/methods may be invoked by the `MAIN` task:

- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`
- `GROUP`(inherited from `PROPERTY-VECTOR`)
- `STATE`(inherited from `PROPERTY-VECTOR`)
- `TIMEOUT`(inherited from `PROPERTY-VECTOR`)
- `GET` (inherited from `PROPERTY-VECTOR`)
- `SEND` (inherited from `PROPERTY-VECTOR`)
- `WAIT` (inherited from `PROPERTY-VECTOR`)
- `SEND&WAIT` (inherited from `PROPERTY-VECTOR`)
- `SEND&OK` (inherited from `PROPERTY-VECTOR`)
- `SEND&IDLE` (inherited from `PROPERTY-VECTOR`)
- `PRINT` (inherited from `PROPERTY-VECTOR`)
- `COPY` (inherited from `INDI-COMPOSITE`)
- `SIZE` (inherited from `INDI-COMPOSITE`)
- `LOOKUP` (inherited from `INDI-COMPOSITE`)

4.2.9 `NUMBER-ITEM` class

The `NUMBER-ITEM` class is the contained class of `NUMBER-VECTOR` It can be directly instantiated.

The following public selectors may be invoked by the `MAIN` task:

- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`
- `MIN-VAL`
- `MAX-VAL`
- `STEP`
- `DISPLAY-FORMAT`
- `SET`
- `GET`
- `COPY`
- `PRINT`
- `SEND`

Public methods

```
m:   ( F: r1 ; S: this -- )           \ overrides set
```

Set the new value ('hold' value) for this `NUMBER-ITEM`.

```
m:  ( this -- F: r1 )                 \ overrides get
```

Get the new 'live' value for this `NUMBER-ITEM`.

```
m:  ( this -- )                       \ overrides print
```

Print `NUMBER-ITEM` name and value to the standard output.

```
m:  ( this -- )                       \ overrides copy
```

Copy 'live' value into 'hold' value.

```
:m min-val   ( this -- ; F -- r1 )
```

Get the item's minimum value. Useful for GUIs.

```
:m max-val   ( this -- ; F -- r1 )
```

Get the item's maximum value. Useful for GUIs.

```
:m step   ( this -- ; F -- r1 )
```

Get the item's step value. Useful for GUIs.

```
:m display-format   ( this -- ca1 u1 )
```

Get the item's suggested print format string. Useful for GUIs.

4.2.10 `LIGHT-VECTOR` class

The `LIGHT-VECTOR` class is the container for all the `LIGHT-ITEM` objects contained within. It can be directly instantiated.

The following public selectors/methods may be invoked by the `MAIN` task:

- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`
- `GROUP`(inherited from `PROPERTY-VECTOR`)
- `STATE`(inherited from `PROPERTY-VECTOR`)
- `GET` (inherited from `PROPERTY-VECTOR`)
- `PRINT` (inherited from `PROPERTY-VECTOR`)
- `SIZE` (inherited from `INDI-COMPOSITE`)
- `LOOKUP` (inherited from `INDI-COMPOSITE`)

INDI Exceptions

```
ErrDef LightVector-Excp "Light vectors can only receive data"
```

Public methods

```
m: ( this --  )                       \ overrides print
```

Print to standard output the property name, state and individual property item names and values.

4.2.11 LIGHT-ITEM class

The LIGHT-ITEM class is the contained class of LIGHT-VECTOR It can be directly instantiated.

The following public selectors may be invoked by the MAIN task:

- NAME (inherited from INDI-ITEM)
- PARENT (inherited from INDI-ITEM)
- LABEL (inherited from INDI-ITEM
- GET
- PRINT

Helper words

`Alias: >$light >$state   ( n -- ca1 u1 )`
The same set of strings and constants are used for property state and light values.

`Alias: >light >state   ( ca1 u1 -- n flag )`
Idem.

public methods

```
  m:   ( this -- n1 )                    \ overrides get
```
Get the new 'live' value for this LIGHT-ITEM.

```
  m:  ( this -- )                        \ overrides print
```
Print LIGHT-ITEM name and value to the standard output.

4.2.12 SWITCH-VECTOR class

The SWITCH-VECTOR class is the container for all the SWITCH-ITEM objects contained within. It can be directly instantiated.

The following public selectors/methods may be invoked by the MAIN task:

- NAME (inherited from INDI-ITEM)
- PARENT (inherited from INDI-ITEM)
- LABEL (inherited from INDI-ITEM
- GROUP(inherited from PROPERTY-VECTOR)
- RULE
- STATE(inherited from PROPERTY-VECTOR)
- TIMEOUT(inherited from PROPERTY-VECTOR)
- GET (inherited from PROPERTY-VECTOR)
- SEND (inherited from PROPERTY-VECTOR)
- WAIT (inherited from PROPERTY-VECTOR)
- SEND&WAIT (inherited from PROPERTY-VECTOR)
- SEND&OK (inherited from PROPERTY-VECTOR)
- SEND&IDLE (inherited from PROPERTY-VECTOR)
- PRINT
- COPY (inherited from INDI-COMPOSITE)
- SIZE (inherited from INDI-COMPOSITE)
- LOOKUP (inherited from INDI-COMPOSITE)

Switch Vector attributes

`0 Constant OneOfMany`
Switch rule value.

`1 Constant AtMostOne`
Switch rule value.

`2 Constant AnyOfMany`
Switch rule value.

`: >$rule   ( n -- ca1 u1)`
Given the permission number *n*, gets its string literal.

`: >rule  ( ca u -- n True | False )`
Translates `"OneOfMany""`, `"AtMostOne"` and `"AnyOfMany""` str. to integer constants `OneOfMany`, `AtMostOne` and `AnyOfMany`. Return true upon succesful conversion.

Public Fields

`cell% field% last-sw                    \ this -- addr`
This field holds the reference to the last switch object changed. This is used by its children `SWITCH-ITEM`s. Usage:

```
 <obj> last-sw @     or   <value> <obj> last-sw !
```

Public methods

`:m rule   ( this -- n1 )`
Get the `switch-vector`'s rule. To be used by the `MAIN` task.

`m: ( this --  )                         \ overrides print`
Print to standard output the property name, state and individual property item names and values.

4.2.13 `SWITCH-ITEM` class

The `SWITCH-ITEM` class is the contained class of `SWITCH-VECTOR` It can be directly instantiated.

The following public selectors may be invoked by the `MAIN` task:

- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`
- `SET`
- `GET`
- `COPY`
- `PRINT`
- `SEND`

Helper words

`: >switch ( ca1 u1 - flag1 True | False )`
Translates `"On"` and `"Off"` INDI strings to boolean flags. *flag1* is the converted value and flag on TOS is the 'found' flag.

`: >$switch   ( flag -- ca1 u1)`
Given the switch *flag*, gets its string literal.

public methods

```
m:   ( flag this -- )         \ overrides set
```
Set the new value ('hold' value) for this `SWITCH-ITEM`.

```
m:   ( this -- flag )         \ overrides get
```
Get the new 'live' value for this `SWITCH-ITEM`.

```
m:  ( this -- )                       \ overrides print
```
Print `SWITCH-ITEM` name and value to the standard output.

```
m:  ( this -- )                       \ overrides copy
```
Copy 'live' value into 'hold' value.

4.2.14 `BLOB-VECTOR` class

The following public selectors/methods may be invoked by the `MAIN` task:

- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`
- `GROUP`(inherited from `PROPERTY-VECTOR`)
- `STATE`(inherited from `PROPERTY-VECTOR`)
- `TIMEOUT`(inherited from `PROPERTY-VECTOR`)
- `GET` (inherited from `PROPERTY-VECTOR`)
- `SET`
- `SEND` (inherited from `PROPERTY-VECTOR`)
- `WAIT` (inherited from `PROPERTY-VECTOR`)
- `SEND&WAIT` (inherited from `PROPERTY-VECTOR`)
- `SEND&OK` (inherited from `PROPERTY-VECTOR`)
- `SEND&IDLE` (inherited from `PROPERTY-VECTOR`)
- `PRINT` (inherited from `PROPERTY-VECTOR`)
- `COPY` (inherited from `INDI-COMPOSITE`)
- `SIZE` (inherited from `INDI-COMPOSITE`)
- `LOOKUP` (inherited from `INDI-COMPOSITE`)

```
property-vector class

   cell% inst-var m-directory           \ directory to save incoming BLOBs

end-class blob-vector
```

```
m:   ( ca1 u1 this -- )                          \  overrides construct
```
Create a new `BLOB-VECTOR` object with name given by *ca1 u1*.

```
m:   ( ca u this -- )                 \ overrides set
```
Set the directory path *ca u* for incoming BLOBs. Create a new directory if does not exists. If no directory is ever set, use the user's current directory. To be used by the `MAIN` task.

```
:m directory@   ( this -- ca u )
```
Get the directory path *ca u* where incoming BLOBs are stored. If *\i{u] is zero, use the current directory. To be used by the `LISTENER` task.

Other helper words

```
: auto-blobs-dir                        \ -- ca u
```

Generate an automatic directory string in the `PAD` with format `$HOME/ccd/<julian day>` to store incoming BLOBs like CCD images. `$HOME` is the user's HOME environment variable. **Note:** This word does **not** create the directory.

4.2.15 `BLOB-ITEM` class

The `BLOB-ITEM` class is the contained class of `BLOB-VECTOR` It can be directly instantiated.

The following public selectors may be invoked by the `MAIN` task:

- `NAME` (inherited from `INDI-ITEM`)
- `PARENT` (inherited from `INDI-ITEM`)
- `LABEL` (inherited from `INDI-ITEM`
- `SET`
- `GET`
- `PRINT`
- `SEND`

Public methods

```
m:   ( ca u this -- )                \ overrides set
```

Set the string with the file name *ca u* for a BLOB to send. Also find out the input file prefix (either normal or .z compressed) and size.

```
m:  ( this -- ca u | x 0 )                \ overrides get
```

Get the file path *ca u* where the received BLOB is stored.

```
m: ( this -- )                       \ overrides send
```

Send BLOB to indiserver and log timestamps.

```
m:  ( this -- )                      \ overrides print
```

Print received `BLOB-ITEM` file name to standard output.

5 Interfaces

5.1 Introduction

The notion of **interface** is well known in programming languages like Java. Interfaces have a name and a set of operations or selectors with a well defined signature. Operation name clashes are solved by the fully qualified operation name, including the interface name.

However, in Forth the situation is somewhat different. There is not an easy way of reusing names other than private wordlists. If two different classes want to use the same name for its public interface, either we choose a synonym or we define this name in a separate `INTERFACE` construct to be overriden conveniently.

The usage of interfaces in `NTOScript` is a bit peculiar in the sense that they are simply collections of related words and underlying concepts like `load` and `save`. However, it is impossible to specify the exact stack signature, because this depends on the class implementing his interface.

5.2 Glossary

5.2.1 Core interfaces

Interfaces defined for the `INDIScript` core.

```
interface
   selector connect                      \ i*x this -- j*x
   selector connected?                   \ i*x this -- j*x
   selector disconnect                   \ i*x this -- j*x
end-interface i-connection


interface
   selector lock                         \ i*x this -- j*x
   selector unlock                       \ i*x this -- j*x
end-interface i-latch

interface
   selector set                          \ i*x this -- j*x
   selector get                          \ i*x this -- j*x
end-interface i-property

interface
   selector push-front                   \ i*x this -- j*x
   selector push-back                    \ i*x this -- j*x
   selector pop-front                    \ i*x this -- j*x
   selector pop-back                     \ i*x this -- j*x
end-interface i-list

interface
   selector next-item                    \ i*x this -- j*x
end-interface i-iterator
```

6 Command Line Interface

An assorted collection of words for command line interface scripting.

6.1 Glossary

6.1.1 Calendar, date & time

Convenient words to express timestamps as date & time to perform busy waiting.

`: JD                \ dd mm yyyy -- jd`
Julian day number, from day *dd*, *mm* and year *yyyy*.

`: seconds                \ +n1 -- +n2`
Convert seconds to milliseconds.

`: minutes                \ +n1 -- +n2`
Convert minutes to milliseconds.

`Alias: waiting  ms                        \ +n1 --`
Wait a number of milliseconds. Calls `PAUSE`. Usage:

```
 34 seconds waiting
  5 minutes waiting
```

Months

`#01 Constant January`

`#02 Constant February`

`#03 Constant March`

`#04 Constant April`

`#05 Constant May`

`#06 Constant June`

`#07 Constant July`

`#08 Constant August`

`#09 Constant September`

`#10 Constant October`

`#11 Constant November`

`#12 Constant December`

`: GMT                                        \ +n1 u2  -- hh mm ss True`
Greenwich Mean Time (or UTC) specifier. *n1 u2* is the sexagesimal number representing time, written as `HH:MM` or `HH:MM:SS`.

`: LT                                         \ +n1 u2 -- hh mm ss False`
Local Time specifier. *n1 u2* is the sexagesimal number representing time, written as `HH:MM` or `HH:MM:SS`.

`: at-time                                    \ yyyy mm dd hh mm ss flag --`
Wait until a given `GMT` or `LT` date&time. Calls `PAUSE`. Usage:

```
   2008 November 11  15:23:30 GMT at-time
   2008 November 11  16:23:30 LT  at-time
```

6.1.2 Script programming constructs

Syntactic sugar to make scritps more readable for newcomers.

`Alias: definition: :`

Start a definition.

`Alias: end-definition ;`

End a definition.

`Alias: fetch @                    \ addr -- n`

Fetch an integer given its address.

`Alias: store !                    \ n addr --`

Store an integer in specified address.

`Alias: 2fetch 2@                  \ addr -- lo hi`

Fetch a double integer starting an given address.

`Alias: 2store 2!                  \ lo hi addr --`

Store a double integer at given address.

`Alias: cfetch c@                  \ addr -- c`

Fetch a character given its address.

`Alias: cstore c!                  \ c addr --`

Fetch a character given its address.

6.1.3 Database scripting words

Database creation starts by creating the top level `INDI-SERVER` object. A convenient parsing word `INDIServer:` is provided. Then, you must connect to the remote server and let the INDI protocol populate the lower levels of the hierarchy. Later on, use state-smart parsing words `DEVICE`, `VECTOR` or `ITEM` to perform database object lookup.

Smart words are evil. Remember not to tick, `[COMPILE]` or `POSTPONE` these words or any word containing them. A paper on the subject may be found at `http://www.complang.tuwien.ac.at/papers/` as ertl98.ps.gz.

Usage of these words assume that a `CurrentServer` is already set. `TheCurrentServer` is set when invoking a `CONNECT` on a given `INDI-SERVER` object. It can also be manually changed by issuing:

```
<an indi-server object> To TheCurrentServer
```

Example of usage:

```
INDIServer: indiserver localhost 7624
indiserver connect
indiserver properties get
2 seconds waiting

device CCDCam print

property vector CCDCam Exposure timeout .

True property item CCDCam Binning 1:1 set

: photo   ( -- )
   5.0 property item   CCDCam ExpValues ExpTime set
       property vector CCDCam ExpValues send&wait
;
```

`: property` \ --

Syntactic sugar for `vector` or `item`. Usage:

```
property vector Camera Exposure ...
property item Camera Exposure ExpDur ....
```

`: properties` \ --

Syntactic sugar for `get`. Usage:

```
indiserver properties get
device Camera properties get
```

`: BLOBs` \ --

Syntactic sugar for usages below:

```
device Camera BLOBs enable
device camera BLOBs disable
```

`: directory` \ --

Syntactic sugar for usage below:

```
s" /tmp" property vector Camera Pixels directory set
```

`: INDIServer:` \ "name" "host" "port" -- ; [child] -- obj

Create a named `INDI-SERVER` object that will connect to host and port given in the input stream. Invoking the name will return the object handle *obj*.

```
INDIServer: indiserver1 localhost 7624
```

`: device` \ "device" -- obj

Find an `INDI-DEVICE` whose name is given in the input stream. Throw a `Dev-Excp` exception if not found.

`: vector` \ "device" "propvector" -- obj

Find a `PROPERTY-VECTOR` whose names are given in the input stream. Can throw a `Dev-Excp` or `Prop-Vector-Excp` exceptions if any object is not found.

`: item` \ "device" "propvector" "propitem" -- obj

Find a `TEXT-ITEM, SWITCH_ITEM, ...` whose names are given in the input stream. Can throw a `Dev-Excp`, `Prop-Vector-Excp` or `Prop-Item-Excp` exceptions if any object is not found.

6.1.4 Angle units

Set of words dediacted to conversions from sexagesimal angles to internal representation (floating point degrees in this case) and to perform formatted output of such quantities.

The idea is to express these quantities in a compact notation and convert to floating point for use `NUMBER-VECTOR`s by using unit suffixes like:

```
 00:42:44 R.A.  +41:16:08 DEC.
+40:25:00 LONG. -03:42:00 LAT.
```

The output format is flexible There are 3 coordinate systems (`EQUATORIAL`, `ALTAZIMUTAL` and (`GEOGRAPHICAL`) and 3 output possible display formats for each one. Examples:

```
standard equatorial   coordinates
lo-res   geographical coordinates
hi-res   altazimutal  coordinates
```

Of course, indiviual words permit complete control of display format in every other situation.

Floating point conversions

`: >fsexag            \ S: n1 u2 -- ; F: -- r1`

Convert a (numerator, denominator pair) *n1 u2* into floating point degrees by dividing n1/u2.

`: >ANGLE                                \ F: r1 -- r2`

Convert a floating point angle *r1* expressed in time units like *Right Ascension* in sexagesimal degrees.

`: >TIME                                 \  F: r1 -- r2`

Convert a floating point angle *r1* expressed in sexagesimal degrees in time units. Used for \i{Right Ascension} angles.

`: R.A.                                  \ n1 u2 -- ; F: -- r1`

Convert a numerator, denominator pair *n1 u2* into floating point *Right Ascension*. Valid range: `00:00:00` to `23:59:59`

`: DEC.                                  \ n1 u2 -- ; F: -- r1`

Convert a numerator, denominator pair *n1 u2* into floating point *Declination*. Valid range: `-90:00:00` to `+90:00:00`

`: LONG.                                 \ n1 u2 -- ; F: -- r1`

Convert a numerator, denominator pair *n1 u2* into floating point *Longitude*. Valid range: `-180:00:00` to `+180:59:59`. East is positive.

`: LAT.                                  \ n1 u2 -- ; F: -- r1`

Convert a numerator, denominator pair *n1 u2* into floating point *Latitude*. Valid range: `-90:00:00` to `+90:00:00`

`: AZ.                                   \ n1 u2 -- ; F: -- r1`

Convert a numerator, denominator pair *n1 u2* into floating point *Azimuth*. Valid range: `+180:00:00` to `-180:00:00`. East of meridian is positive

`: ALT.                                  \ n1 u2 -- ; F: -- r1`

Convert a numerator, denominator pair *n1 u2* into floating point *Altitude*. Valid range: `-90:00:00` to `+90:00:00`

`: DEG.                    \ n1 u2 -- ; F: -- r1`
Convert a numerator, denominator pair *n1 u2* into floating point *Degrees*.

Angles formatted printing

`: .DD:MM                  \ F: r1 --`
Print sexagesimal angle *\{r1} as `+DD:MM`.

`: .DDD:MM                 \ F: r1 --`
Print sexagesimal angle *\{r1} as `+DDD:MM`.

`: .DD:MM.M                \ F: r1 --`
Print sexagesimal angle *\{r1} as `+DD:MM.M`.

`: .DDD:MM.M               \ F: r1 --`
Print sexagesimal angle *\{r1} as `+DDD:MM.M`.

`: .DD:MM:SS               \ F: r1 --`
Print sexagesimal angle *\{r1} as `+DD:MM:SS`.

`: .DDD:MM:SS              \ F: r1 --`
Print sexagesimal angle *\{r1} as `+DDD:MM:SS`.

`: .DD:MM:SS.S             \ F: r1 --`
Print sexagesimal angle *\{r1} as `+DD:MM:SS.S`.

`: .DDD:MM:SS.S            \ F: r1 --`
Print sexagesimal angle *\{r1} as `+DDD:MM:SS.S`.

`Defer (.DEG)             \ F:  r1 --`
Sexagesimal format Generic Degrees printing routine.

`Defer (.RA)              \ F:  r1 --`
Sexagesimal format Right Ascension printing routine.

`Defer (.DEC)             \ F:  r1 --`
Sexagesimal format Declination printing routine.

`Defer (.AZ)              \ F:  r1 --`
Sexagesimal format Azimuth printing routine.

`Defer (.ALT)             \ F:  r1 --`
Sexagesimal format Altitude printing routine.

`Defer (.LONG)            \ F:  r1 --`
Sexagesimal format Longitude printing routine.

`Defer (.LAT)             \ F:  r1 --`
Sexagesimal format Latitude printing routine.

`: .DEG                     \ F: r1 --`
Print a floating point *r1* as sexagesimal format plus `R.A.` label.

`: .RA                      \ F: r1 --`
Print a floating point *r1* as sexagesimal format plus `R.A.` label.

`: .DEC                     \ F: r1 --`
Print a floating point *r1* as sexagesimal format plus `DEC.` label.

`: .LAT \ F: r1 --`
Print a floating point *r1* as sexagesimal format plus `LAT.` label.

`: .LONG \ F: r1 --`
Print a floating point *r1* as sexagesimal format plus `LONG.` label

`: .AZ \ F: r1 --`
Print a floating point *r1* as sexagesimal format plus `AZ.` label.

`: .ALT \ F: r1 --`
Print a floating point *r1* as sexagesimal format plus `ALT.` label.

Angles formatted output

`0 Constant Equatorial`
Equatorial coordinates printing words `.RA` and `.DEC`

`1 Constant Altazimutal`
Altazimutal coordinates printing words `.AZ` and `.ALT`

`2 Constant Geographical`
Geographical coordinates printing words `.LAT` and `.LONG`
Display resolution constants

`0 Constant hi-res`
Select the highest resolution for the selected set of coordinates.

`1 Constant standard`
Select the most commonly used resolution for the selected set of coordinates.

`2 Constant lo-res`
Select the lowest resolution for the selected set of coordinates.

`: coordinates \ resol system --`
Select the resolution for a given set of coordinates and resolution. Example:

```
standard equatorial coordinates
```

6.1.5 Other Physical/Non-physical Units

Unit names as suffixes to make scripts more readable when setting property values. These units are meant to be used when writting `NUMBER-VECTOR`s whose values are `always` floating point.

Floating point Units

To use these units, preceeding numbers **must** include a dot.

`: sec. \ d1 -- ; F: -- r1`
Time seconds. Can express fractions of seconds.

`: e-/ADU \ d1 -- ; F: -- r1`
For GAIN conversion word. Usage:

```
2.5 e-/ADU
```

`: e- \ d1 -- ; F: -- r1`
For RDNOISE conversion word. Also can express fractional electrons coming from statistical estimation. Usage:

```
34. e-
```

`: arcsec/pixel \ d1 -- ; F: -- r1`

SCALE conversion word. Usage:

```
2.3 arcsec/pixel
```

`: volts \ d1 -- ; F: -- r1`
VPELT conversion word. Volts can be fractional. Usage:

```
12. volts
```

`: degrees \ --`
Syntactic sugar for the word below.

`: celsius \ d1 -- ; F: -- r1`
HOT,COLD temperature conversion word. Usage:

```
-30. degrees celsius
```

`: cm. \ d1 -- ; F: -- r1`
Convert *r1* centimeters to *r2* meters. Usage:

```
100. cm.
```

`: mm. \ d1 -- ; F: -- r1`
Convert *r1* milimeters to *r2* meters. Usage:

```
3. mm.
```

`: um. \ d1 -- ; F: -- r1`
Convert *r1* microns to *r2* meters. Usage:

```
9. um.
```

`: inch. \ d1 -- ; F: -- r1`
Convert *r1* inches to *r2* meters. Usage:

```
9. inch.
```

Integer Units

To use these units, preceeding numbers **must not** include a dot.

`Alias: units s>f \ n1 -- ; F: -- r1`
Express generic integer quantities as floating point. Usage:

```
12 units
```

`Alias: images s>f \ n1 -- ; F: -- r1`
Number of images as floating point. Usage:

```
4 images
```

`Alias: pixels s>f \ n1 -- ; F: -- r1`
Express pixels as floating point. Usage:

```
234 pixels
```

`Alias: buffers s>f \ n1 -- ; F: -- r1`
Express pixels as floating point. Usage:

```
234 pixels
```

`Alias: #times s>f \ n1 -- ; F: -- r1`
Express number of times as floating point. Usage:

```
234 #times
```

`: x \ --`
Syntactic sugar to express pair of things like dimensions. Usage:

```
6.0 mm. x 4.0 mm.
```

7 Sky Mapping

7.1 Introduction

The sky mapping package provides a tool to generate (R.A., DEC.) coordinate pairs that sample 'rectangular' areas in the sky for various tasks like mosaics or asteroid hunting. It includes the necessary declination correction to account for the noticeable decreasing RA arc length produced at medium to high declinations.

Coordinates generation (sampling) can be according to the following patterns:

- `UNIFORM` pattern
- `ZIGZAG` pattern

```
>---------*      +---------*
                 |
>----x---->      +----x----+
                           |
o--------->      o---------+
```

and each pattern in different modes, which account for the differenct combinations: `+RA +DEC`, `-RA -DEC`, `+RA -DEC` and `-RA +DEC`. The fast moving axis is always RA.

For a given set of initial and final (RA,Dec) coordinates, there are two possibilities for mosaicing the zone. To resolve this ambiguity we have introduced a limitation. The mosaicing zone should not cover more than 12:00:00 hours in R.A. The mapping algorithm always choose the smaller zone.

7.1.1 Declination correction

In a spherical coordinates system such as the celestial RA, Dec system, CCD frames sample arc lengts, not central angles and RA is the central angle. It is well known that this relationship follows the following formulae

```
RA arc length = RA * cos(Dec)
```

If we wish to maintain the same RA arc length at different declinations as we move up and down, we need to introduce such correction.

7.1.2 Creating sky mapper objects

These objects are created with the flags given above and the `SkyMapper:` word. Examples:

```
+RA +DEC UNIFORM SkyMapper: skymapper1
+RA -DEC ZIGZAG  SkyMapper: skymapper2
```

Once the sky-mapper is created, we must initialize it with the desired sky area. There are two ways, either specify an initial and final coordinate pairs or specify a central point plus an amount of CCD frames for each coordinate. The specified coordinates always refer to the field centre. The CCD frame size **in degrees** must also be specified for both methods.

```
 00:50:00 R.A.  +40:00:00 DEC. skymapper1 set-initial
 00:35:00 R.A.  +43:00:00 DEC. skymapper1 set-final
 01:00:00 DEG.   01:00:00 DEG. skymapper1 set-frame
```

```
00:45:22 R.A.   85:15:00 DEC. skymapper2 set-centre
                5 x 5 frames  skymapper2 set-frames
01:00:00 DEG.   01:00:00 DEG. skymapper2 set-frame
```

We can also specify and additional overlap factor.

```
25 %RA  25 %DEC skymapper1 set-overlap
```

The overlap factor is 0% for no overlap and 100% for full overlap (causing, in fact, an infinite loop.). Negative values create gaps between frames.

Once all these parameters are set, we can test the produced coordinates by typing:

```
skymapper1 print
```

The two different input methods account for two different needs:

1. **Mosaicing**. Use initial and final points, estimated from the object size as seen in a planetarium program.
2. **Asteroid hunting**. Use field centre plus additional CCD frames around the estimated position.

They produce slightly different results. The former introduces a hard limit in the inital and final coordinates introduced. The second one calculates the whole RA arcs needed at the mid point with the given number of CCD frames and this arc is maintained up un down the RA strippes.

7.1.3 XEphem tool

It is possible to graphically depict what is the mapped area, each one of the frames and the sequence order with XEphem. The tool also uses the predefined Linux FIFOs `'/usr/local/xephem/fifos/xephem_in_fifo'` and `'/usr/local/xephem/fifos/xephem_loc_fifo'` for interactive communication.

Word `XEphem-Tool:` creates a named tool object.

The tool generates two files:

- A eyepieces location file named `'mapping.epl'` and
- An annotations file named `'mapping.ano'`, with the sequence numbers.

These two files are placed under the user's `'$HOME/.xephem'` directory.

To generate the files, we must first assign it to a previous sky mapper object and invoke method `generate`. Example:

```
foreground logging on
XEphem-Tool: xephem
skymapper1 xephem set-mapper
xephem generate

2008-04-20T23:06:46: 10 samples for eyepieces & annotations.
```

Pay attention to the script output, as it tells the number of frames generated to map the desired sky area.

Now, launch XEphem and select your favourite SkyView from the History menu. Then go to the mapped area by manually scrolling or using XEphem data index facility or using again `NTOScript`.

For this latter:

1. Make sure XEphem is not connected to an INDI server.
2. `'Telescope > Configure... > Enable sky marker from'`
3. `'Telescope > Keep visible'`
4. and then type at the `NTOScript` console: `@`fo`{`xephem recenter `}`.

A marker should be displayed on the approximate center of the mapped area. Then, load the eyepieces and annotation files as usually done in XEphem: `'Control > Eyepieces...'` and `'Control > User annotation...'` lays in between.

7.2 Glossary

7.2.1 `SKYMAPPER` module

This module enapsulate the creation of the appropiate sky mapper object given the initial flags. Three flags must be given:

1. The way the R.A. coordinate should vary: `+RA`, `-RA`
2. The way the Dec. coordinate should vary: `+DEC`, `-DEC`
3. The mapping pattern: `UNIFORM`, `ZIGZAG`.

Creation flags

`0 Constant Uniform`
Select the uniform pattern.

`1 Constant ZigZag`
Select the zigzag pattern.

`$00 Constant -RA`
R.A. coordinate should decrease.

`$01 Constant +RA`
R.A. coordinate should increase.

`$00 Constant -DEC`
Dec. coordinate should decrease.

`$02 Constant +DEC`
Dec. coordinate should increase.

`: (SkyMapper)                          \ raflag decflag pattern -- obj`
Create an anonymous sky mapper object based on the *pattern*, *\{rafflag} and *decflag* creation flags passed.

`: SkyMapper:          \ raflag decflag pattern "name" -- ; [child] obj`
Create a named sky mapper object based on the *pattern*, *\{rafflag} and *decflag* passed.

7.2.2 /EQPOINT structure

```
struct /EqPoint
```
Equatorial coordinates structure.

```
   1 floats field sky.ra               \ Right Ascension
   1 floats field sky.dec              \ Declination
end-struct
```

`: EqPoint!   ( F: dec ra -- ; S: p1 --  )`
Write coodinate pairs on the float stack to the `/EqPoint` structure. Reverse ordering of coordinates ensures easy access to *ra* and peform conversions from time to degrees.

`: EqPoint@   ( F: -- dec ra ; S: p1 --  )`
Read coodinate pairs from the `/EqPoint` structure to the float stack. Reverse ordering of coordinates ensures easy access to *ra* and peform conversions from degrees to time.

`: initEqPoint                              \ p1 -- ; F: ra dec --`
Initialize a `/EqPoint` structure *p1*, with *ra* and *dec* initial values.

`: EqPoint:                       \ "name" ; F: ra dec -- ; [child] addr`
Create a `/EqPoint` named structure in the dictionary.

`: .2xDEG                                   \ F: ra dec --`
Print a pair of floating point coordinates both in degrees where *ra* can be Right ascension or azimith, and *dec* may be Declination or Altitude.

`: .RA.DEC                                  \ F: ra dec --`
Print to standard output the RA & DEC float coordinates. There is no conversion for RA and will print whatever angle value is passed.

`: .EqPoint                                 \ p1 --`
Print to standard output the RA and DEC coordinates of a *p1* `/EqPoint` structure.

7.2.3 /GEOPOINT structure

```
struct /GeoPoint
```
Geographical coordinates structure.

```
   1 floats field geo.long             \ Longitude in degrees
   1 floats field geo.lat              \ Latitude  in degrees
end-struct
```

`: GeoPoint!   ( F: long lat -- ; S: p1 --  )`
Write coodinate pairs on the float stack to the `/GeoPoint` structure.

`: GeoPoint@   ( F: -- long lat ; S: p1 --  )`
Read coodinate pairs from the `/GeoPoint` structure to the float stack.

`: initGeoPoint                          \ p1 -- ; F: long lat --`
Initialize a `/GeoPoint` structure *p1*, with *long* and *lat* initial values.

`: GeoPoint:                   \ "name" ; F: long lat -- ; [child] addr`
Create a `/GeoPoint` named structure in the dictionary.

`: .LONG.LAT                             \ F: long lat --`
Print to standard output the long and lat coordinates.

`: .GeoPoint                             \ p1 --`
Print to standard output the RA and DEC coordinates of a *p1* `/GeoPoint` structure.

7.2.4 `GRID-GENERATOR` class

Abstract class to factor common code and additional interface for all grid-style sky point generators These grid-style sky point generators sweep the sky in bands, either in RA or DEC order, ascending or descending, uniformingly or performing zig-zags.

All RA coordinates in this class are stored in degrees, for convenience.

Public methods

`m:   ( this -- )                        \ overrides construct`
Default constructor.

`:m set-initial   ( F: ra dec -- ; S:  this --  )`
Set the initial Sky Point. Convert *ra* to degrees.

`:m set-final   ( F: ra dec -- ; S:  this --  )`
Set the final Sky Point. Convert *ra* to degrees.

`:m set-centre   ( F: ra dec -- ; S:  this --  )`
Set the center Sky Point in the centre + frames approach. Convert *ra* to degrees.

`:m set-frame   ( F: ra dec -- ; S:  this --  )`
Set the increment both in *ra* and *dec*. Both magnitudes must be previously in degrees.

`:m set-frames   ( n1 n2 this --  )`
Set the number of frames in RA (*n1*) and DEC (*n2*) to take. *n1* and *n2* must be odd numbers. Used in the centre + frames approach, must be called **after** `set-frame` and `set-centre`.

`:m set-overlap   ( n1 n2 this -- )`
Set the user defined overlap factors that will be applied to both frame axis in RA and Dec respectively. *n1* is the `%RA` overlap and *n2* is the `%DEC` overlap. Range for all: 0 `<=` `|n|` `<` 100 . Negative values in fact leave holes.

`m:   ( this -- )                        \ overrides print`
Iterates through the initial until final sky points, printing to stdout the (RA, Dec) coordinates.

`:m seq#  ( this -- n )`
Obtains the sequence number for the current point. Must be called after`next-point`. To be used by auxiliar tools.

```
:m frame   ( this -- addr )
```

Get the frame size *fo{/SkyPoint} structure. To be used by auxiliar tools.

```
:m initial   ( this -- addr )
```

Get the initial *fo{/SkyPoint} structure. To be used by auxiliar tools.

```
:m final   ( this -- addr )
```

Get the final *fo{/SkyPoint} structure. To be used by auxiliar tools.

7.2.5 `RA+DEC+UNIFORM-RA` class

(+RA, +DEC) increments, RA bands, uniform sky mapper.

Public methods

```
m:   ( this -- )                          \ overrides reset
```

Reset internal state to start a new iteration.

```
m:   ( this -- p1 True | False)       \ overrides next-point
```

Get next sky point *p1* or a false flag when finished.

7.2.6 `RA-DEC-UNIFORM-RA` class

(-RA, -DEC) increments, RA bands, uniform sky mapper

Public methods

```
m:   ( this -- )                          \ overrides reset
```

Reset internal state to start a new iteration.

```
m:   ( this -- p1 True | False)       \ overrides next-point
```

Get next sky point *p1* or a false flag when finished.

7.2.7 `RA-DEC+UNIFORM-RA` class

(-RA, +DEC) increments, RA bands, uniform sky mapper

Public methods

```
m:   ( this -- )                          \ overrides reset
```

Reset internal state to start a new iteration.

```
m:   ( this -- p1 True | False)       \ overrides next-point
```

Get next sky point *p1* or a false flag when finished.

7.2.8 `RA+DEC-UNIFORM-RA` class

(+RA, -DEC) increments, RA bands, uniform sky mapper

Public methods

```
m:   ( this -- )                          \ overrides reset
```

Reset internal state to start a new iteration.

```
m:   ( this -- p1 True | False)      \ overrides next-point
```

Get next sky point *p1* or a false flag when finished.

7.2.9 `RA+DEC-UNIFORM-RA` class

(+RA, -DEC) increments, RA bands, uniform sky mapper

Public methods

```
m:   ( this -- )                         \ overrides reset
```
Reset internal state to start a new iteration.

```
m:   ( this -- p1 True | False)      \ overrides next-point
```
Get next sky point *p1* or a false flag when finished.

7.2.10 ZIGZAG_GENERATOR class

Abstract class for all zigzag sky mappers

Public methods

```
m:   ( this -- p1 True | False)      \ overrides next-point
```
Get next sky point *p1* or a false flag when finished.

7.2.11 RA+DEC+ZIGZAG-RA class

(+RA, +DEC) increments, RA bands, zig-zag sky mapper.

Public methods

```
m:   ( this -- )                         \ overrides reset
```
Reset internal state to start a new iteration.

7.2.12 RA-DEC-ZIGZAG-RA class

(-RA, -DEC) increments, RA bands, zig-zag sky mapper.

Public methods

```
m:   ( this -- )                         \ overrides reset
```
Reset internal state to start a new iteration.

7.2.13 RA-DEC+ZIGZAG-RA class

(-RA, +DEC) increments, RA bands, zig-zag sky mapper.

Public methods

```
m:   ( this -- )                         \ overrides reset
```
Reset internal state to start a new iteration.

7.2.14 RA+DEC-ZIGZAG-RA class

(+RA, -DEC) increments, RA bands, zig-zag sky mapper.

Public methods

```
m:   ( this -- )                         \ overrides reset
```
Reset internal state to start a new iteration.

7.2.15 FP Stack operations

```
2.0e0 FConstant %2                       \ -- f#(2.0)
```
Floating Point 2.0

```
: f2dup                          \  F: r1 r2 -- r1 r2 r1 r2
```

Floating point equivalent of `2DUP`

```
: frot-                          \ F: r1 r2 r3 -- r3 r1 r2
```

Floating point equivalent of `-ROT`

```
: -freduce-sg                \ F: r1 -- r2
```

Reduce negative sexagesimal floating point value *ri* to 0..360.0 range.

```
: +freduce-sg                \ F: r1 -- r2
```

Reduce positive sexagesimal floating point value *ri* to 0..360.0 range.

```
: freduce-sg                             \ F: r1 -- r2
```

Reduce positive or negative sexagesimal floating point value *ri* to 0..360.0 range.

7.2.16 Conversion Units

```
: %RA                                    \ n1 -- n1
```

Overlap Percentage of RA. Range: 0 <= u1 < 100

```
: %DEC                                   \ n1 -- n1
```

Overlap Percentage of DEC. Range: 0 <= u1 < 100

```
: frames                                 \ --
```

Expreses number for frames. Syntactic sugar. Intended for use with `set-frames` method.

```
  3 x 4 frames  skymapper2 set-frames
```

7.2.17 CCD and OTA Instrument parameters

Structures to hold instrimental data for various tools and purposes like calculating the FOV of a given CCD Chip through a given OTA.

```
struct /CCDChip
```

CCD Chip structure

```
   1 floats field ccd.width            \ whole imaging width in meters
   1 floats field ccd.height           \ whole imaging area  in meters
   1 floats field ccd.wpix             \ pixel phyisical width in meters
   1 floats field ccd.hpix             \ pixel phyisical height in meters
   1 cells  field ccd.wres             \ width resolution
   1 cells  field ccd.hres             \ height resolution
end-struct
```

```
: CCDChip:      \ S: resw resh "name" -- ; F: wpix hpix -- ; [child] -- addr
```

Create a `/CCDChip` structure with given resolution *resw* and *resw* in pixels and pixel phisical width *wpix* and *hpix*.

```
    1024 x 768  9.0 um. x 9.0 um. CCDChip: KAF-400
```

```
struct /OTA
```

Optical Tube Assembly structure.

```
   1 floats field ota.focal            \ in meters
   1 floats field ota.diam             \ in meters
end-struct
```

```
: OTA:                     \ "name" ; F: focal diam -- ; [child] -- addr
```

Create an `/OTA` structure with a given focal length and diameter.

```
80.0 inch. x 8.0 inch. OTA: LX200-8"@f/10
```

`: FOV                                    \ ota ccd -- width height`

Calculates the FOV in floating point degrees of a *ccd* chip through a given *ota*. The resulting *width* angular dimension is usually assigned to RA and the *height* to Dec.

7.2.18 `XEPHEM-TOOL` class

This class is reponsible to interface XEphem to interactively define the initial and final sky mapping points and generate the *'mapping.epl'* and *'mapping.ano'* files for the user to graphically display the mapped fields and sequence order.

Auxiliary words

Structure

`m: skip-field  ( ca1 u1 this -- ca2 u2 )`

Skip one field altogether *ca1 u1* and get next field *ca1 u1*.

Public methods

`m:   ( this -- )                         \ overrides construct`

Class constructor,

`:m set-mapper    ( mapper this -- )`

Set the proper `GRID-GENERATOR` sky mapper object instance.

`:m generate                              \ --`

Generate *'mapping.epl'* and *'mapping.ano'* files for the user to display the mapped zones and order.

`:m capture                               \ --`

Recenter graphical field on XEphem's Sky View. Receive the initial and final points by graphically clicking on XEphem's Receive 2 points from XEphem SkyView and generate *'mapping.epl'* and *'mapping.ano'* files for the user to display the mapped zones and order.

`: XEphem-Tool:                           \ "name"`

Create a named object of class `XEPHEM-TOOL`

8 Sexagesimal Numbers

8.1 Introduction

This package defines words for sexagesimal number input and conversions. Sexagesimal input may take one of the following forms:

1. `+d:mm`
2. `+d:mm.m`
3. `+d:mm:ss`
4. `+d:mm:ss.s`

The output of such conversions is a double cell number which represent numerator and denominator pairs. The individual numbers are in the units of the least significant digit introduced, that is:

- minutes (denominator = 60)
- 1/10ths of minutes (denominator = 600)
- seconds (denominator = 3600)
- 1/10ths of seconds (denominator = 36000)

Formatted output back to its original string is straightforward when sexagesimal numbers stay in this canonical form. For others, a normalization process may take place to the preferred format, lets say degrees, minutes and seconds, (denominator = 3600).

Fractions of minutes or seconds are recoginzed using the current `FP-CHAR` setting (VFX default setting is `.`). Fractions of degrees are really floating point numbers and as such they are forwarded to the old handler. The `DPL` user variable is set upon recognizing those above. these exceptions is also available, but with the overhead of setting up the exception stack.

8.2 Glossary

```
MODULE SexagPack                        \ sexagesimal numbers package
```

8.2.1 Input conversion

```
: i10**n                                 \ +n1 -- 10**n1
```

Integer only version of `10**n`. *n1* range from 0..9

```
: ($>sexag)                          \ ca1 u1 -- +n2 u2 2 | ca1 u1 0
```

Convert sexagesimal formatted input string *ca1 u1* into a numerator denominator pair *n2 u2*. Numerator is signed. If conversion is not possible, return 0 and input string. Conversion fails for pure integers, doubles and floats. If illegal characters are present, throw `INVALID_NUMERIC_ARG` exception. Leading blanks in *ca1 u1* are counted as illegal characters.

```
: $>sexag                    \ ca1 u1 -- n2 u2 2 | ca1 u1 0
```

Convert input, sexagesimal formatted, counted string *ca1 u1* into a numerator denominator pair *n2 u2*. Numerator is signed. If conversion is not possible, return 0 and input string. Conversion fails for pure integers, doubles and floats. Catches `INVALID_NUMERIC_ARG` exceptions and turn them into failed cases.

```
: sexag-number?                 \ c-addr -- 0 | n 1 | d 2 | -2 F: -- r
```

Convert input, sexagesimal formatted, counted string *ca1 u1* into a numerator denominator pair *d*. Numerator is signed. Denominator is the high cell of *d*. Conversion integers, doubles and floats are forwarded to the old handler. If conversion fails, return 0.

`: sexagesimal                                        \ --`
Turn on sexagesimal input number conversion

`: no-sexagesimal                                     \ --`
Turn off sexagesimal input number conversion

8.2.2 Formatted output

`: <.D:MM>                              \ n1 +n2 --`
Print sexagesimal number identified by its numerator *n1* in minutes (assume denominator 60) as `+D:MM`. *n2* is the right justification for the degrees part (1..3). Padding is inserted using zeroes.

`: <.D:MM.M>                            \ n1 +n2 --`
Print sexagesimal number identified by its numerator *n1* in 1/10th of minutes (assume denominator 600) as `+D:MM.M`. *n2* is the right justification for the degrees part (1..3). Padding is inserted using zeroes.

`: <.D:MM:SS>                           \ n1 +n2 --`
Print sexagesimal number identified by its numerator *n1* in seconds (assume denominator 3600) as `+D:MM:SS`. *n2* is the right justification for the degrees part (1..3). Padding is inserted using zeroes.

`: <.D:MM:SS.S>                         \ n1 +n2 --`
Print sexagesimal number identified by its numerator *n1* in 1/10th of seconds (assume denominator 36000) as `+D:MM:SS`. *n2* is the right justification for the degrees part (1..3). Padding is inserted using zeroes.

`: >cansex                              \ n1 u2 -- n2 3600`
Normalize any sexagesimal number *n1 u2* to canonical form *n2 3600*. Perform internal rounding. Used to print sexagesimal numbers to a default format `+D:MM:SS`.

`: .sexag            \ n1 u2 --`
Formatted printing of sexagesimal number*n1 u2* .

9 Object Oriented Programming

9.1 Introduction

This module is a porting of GForth's `'objects.fs'` object oriented package. As such, it maintains all the features found in the original package. However, many of the internal words have been hidden inside the module and a 'persistant' extension have been made. Please consult the GForth User's manual for an introduction on how to use this package.

9.1.1 Summary of changes

1. **Modifying the Gforth's catch**. VFX Forth uses a separate exception stack, so >ep and ep> are used, instead or >r and >r.
2. **Deleted some helper words**. Words like `-rot`, `?dup-if` or `\g` have been deleted.
3. **Added word**: `[parent']` to help catching exceptions in methods when calling the parent method for a given selector.
4. **User defined exception**: `Method-Excp` instead of `abort"` for missing selector implementation.
5. **Thread-safe implementation of the 'this' pointer**. `this` is no longer a `VALUE` but a `USER` variable.
6. **Added persistance support for turnkey file generation**
7. `Construct`, a non documented VFX word has been redefined and adopted as the constructor. No side effects have been observed.

9.1.2 Exceptions

Added Anton's new word [parent'] to get the xt of the parent method. This allows catching exceptions within methods.

instead of:

```
this [parent]  foomethod
```

to capture an exception, write:

```
this [parent'] foomethod catch
```

9.1.3 Adding Persistance

A noteworthy feature absence is the lack of persistance support for saved images or turnkey applications. Indeed, the class/interface maps are stored in the heap, so a new turnkey application will not have them and will crash.

To alleviate this, a `persistant` word has been defined. It is automatically invoked at `end-interface`. However, it must be manually invoked after `end-methods` or `end-class`, depending on your class declaration style. If you want to generate turnkey images, you should use it in each and every class you define. If you always stick to the same declaration style in your project you can make it easy with a redefinition, for instance:

```
: end-class    end-class persistant ;
```

These data structures are interbally defined using the ported `GForth-Struct` module.

The `interface%` structure is allocated in the dictionary. (aka objects). This structure is also allocated in the dictionary.

This is where the real XT's of methods are stored, Both interfaces

Selectors are structures compiled in the dictionary having the following elements:

`selector-offset` is the possitive offset within the interface-map that retrieves the real XT to execute. `selector-interface` is the interface unique id that this selector belongs to. As we know, this is alos the offset to retrieve an interface-map for a given class-map.

9.2 Glossary

`ErrDef Method-Excp "No method defined for this object/selector combination"`

`: selector ( "name" -- )`
Create selector "*name*" for the current class and its descendents; you can set a method for the selector in the current class with `overrides`.

`: overrides ( xt "selector" -- ) \ objects- objects`
replace default method for "*selector*" in the current class with *xt*. `overrides` must not be used during an interface definition.

9.2.1 Adding persistance

`: persistant \ --`
Copy the last defined interface map from the heap back to the dictionary so that it can be safely saved in turnkey applications.

9.2.2 Interfaces

`Variable last-interface-offset`
Every interface gets a different offset; the latest one is stored here.

`: interface ( -- )`
Start an interface definition.

`: end-interface ( "name" -- )`
End an interface definition and give it a "*name*".

9.2.3 Vsibility control

`: protected ( -- ) \ objects- objects`
Set the compilation wordlist to the current class's wordlist.

`: public ( -- ) \ objects- objects`
Restore the compilation wordlist that was in effect before the last `protected` that actually changed the compilation wordlist.

9.2.4 Classes

`: methods ( class -- )`
Makes *class* the current class. This is intended to be used for defining methods to override selectors; you cannot define new fields or selectors.

`: class ( parent-class -- align offset )`
Start a new class definition as a child of *parent-class*. *align offset* are for use by `field%` etc.

`: end-methods ( -- )`
Switch back from defining methods of a class to normal mode (currently this just restores the old search order).

`: end-class ( align offset "name" -- )`
End a class definition and give it a name.

9.2.5 Classes that implement interfaces

`: implementation ( interface -- )`
The current class implements *interface*. I.e., you can use all selectors of the interface in the current class and its descendents.

9.2.6 this/self, instance variables etc.

`: this ( -- object )`
Thread-safe implementation of 'this'

`: to-this ( object -- )`
Thread-safe implementatio to set a new 'this' pointer

`: m: ( -- xt colon-sys; run-time: object -- )`
Start an anonymous method definition; *object* becomes new `this`. Has to be ended with `;m`.

`: :m ( "name" -- xt; run-time: object -- )`
Start a named method definition; *object* becomes new `this`. Has to be ended with `;m`.

`: exitm ( -- ) \ objects- objects`
`exit` from a method; restore old `this`.

`: ;m ( colon-sys --; run-time: -- )`
End a method definition; restore old `this`.

`: inst-var ( align1 offset1 align size "name" -- align2 offset2 )`
Define a private variable with a given *align and size* and name.

`: inst-value ( align1 offset1 "name" -- align2 offset2 )`
Define a private `VALUE` with a given name.

`: [to-inst] ( compile-time: "name" -- ; run-time: w -- )`
Store *w* into field *name* in `this` object.

9.2.7 Exception handling

9.2.8 Class binding stuff

9.2.9 The `OBJECT` root class

`method construct ( ... object -- )`
Initialize the data fields of *object*. The method for the root class `object` just does nothing.

`method print ( object -- )`
Print the object. The method for the root class `object` prints the address of the object and the address of its class structure.

`selector equal ( object1 object2 -- flag )`
Default object comparison. Should be overriden by subclasses.

`end-class object ( -- class ) \ objects- objects`
the ancestor of all classes.

9.2.10 Constructing objects

`: dict-new ( ... class -- object )`
`allot` and initialize an object of class *class* in the dictionary. `allocate` and initialize an object of class *class*.

10 Support Structs Package

10.1 Introduction

This module is a porting of GForth's ANS structures. It is **not** intended as a replacement for VFX structures, but as the necesary support for the object oriented package also ported from GForth. In a future, it could be possible to rewrite the object oriented package to use the native VFX structures and this module would dissapear. Also, there is an upcoming structures standard in Forth 200x that could be taken into account.

To use this package, it is necessary to previously include a floating point package from those available in VFX or else redefine words `dfaligned`, `sfaligned`, `faligned`, `floats`, `sfloats`, `dfloats`.

The necessary words that specify field size when defining class field members are:

- `cell%`
- `double%`
- `char%`
- `float%`
- `sfloat%`
- `dfloat%`

When defining an array of , let's say three floats, use the expression:

```
float% 3 *
```

When accomodating an VFX structure named `/foo`, use:

```
char% /foo *
```

In addition, when defining public field members, the word `field%` must be used.

10.2 Glossary

`: create-field ( align1 offset1 align size "name" --  align2 offset2 )`

Creates an struct field given by "*name*" with the desired *align* and *size* specification and taking the past history of *align1* and *offset1* to produce the new *align2* and *offset2*.

`: field% ( align1 offset1 align size "name" --  align2 offset2 )`

Defining word for fields, using `create-field` above and the runtime code given by `dofield` and `dozerofield` The defined chilkd word has the following stack signature: `addr1 -- addr2`.

`: end-struct% ( align size "name" -- )`

Finish the whole structure definition and and tag it with "name".

`1 chars 0 end-struct% struct%           \ --  align1 0`

Start a new structure definion. Returns the initial aligment of 1 chars and an offset of 0.

10.2.1 Field size specifiers

```
1 aligned   1 cells   2constant cell%   \ -- align size
```
Define a cell field.

```
1 chars     1 chars   2constant char%   \ -- align size
```
Define a char field.

```
1 faligned  1 floats  2constant float%  \ -- align size
```
Define a float field.

```
1 dfaligned 1 dfloats 2constant dfloat% \ -- align size
```
Define a double precision floating point field.

```
1 sfaligned 1 sfloats 2constant sfloat% \ -- align size
```
Define a single precision floating point field.

```
cell% 2*              2constant double% \ -- align size
```
Define a double cell number field.

10.2.2 Memory allocation words

Memory allocation words for these structures either in the dictionary or in the heap.

```
: %allot   ( align size -- addr )
```
Returns an aligned memory region *addr* of a given *size* in the dictionary with the respect to *align* bytes.

```
: %alloc   ( align size-- addr )
```
Returns an aligned memory region *addr* of a given *size* in the heap with the respect to *align* bytes. Throws a `#-59` exception code if allocation fails.