

# Catalina



**Catalyst  
Operating System  
Reference Manual**

**Release 3.3**

## Table of Contents

|   |           |
|---|-----------|
| <b>WHAT IS CATALYST? .....</b>                            | <b>4</b>  |
| <b>STATUS.....</b>  | <b>4</b>  |
| <b>FEATURES .....</b>                                     | <b>4</b>  |
| <b>LICENSE.....</b>                                       | <b>5</b>  |
| <b>INSTALLING CATALYST.....</b>                           | <b>5</b>  |
| <b>OVERVIEW.....</b>                                      | <b>5</b>  |
| <b>INSTALLING THE CATALYST BINARY RELEASE.....</b>        | <b>5</b>  |
| <i>Load the Catalyst binary into EEPROM.....</i>          | <i>5</i>  |
| <i>Load the Catalyst programs onto an SD Card.....</i>    | <i>6</i>  |
| <b>INSTALLING THE CATALYST SOURCE RELEASE.....</b>        | <b>7</b>  |
| <i>Catalyst Directory Structure.....</i>                  | <i>7</i>  |
| <i>Building Catalyst from Source.....</i>                 | <i>8</i>  |
| <i>Compiling Programs for Catalyst.....</i>               | <i>9</i>  |
| <i>Compiling C programs to run under Catalyst.....</i>    | <i>9</i>  |
| <i>Compiling SPIN programs to run under Catalyst.....</i> | <i>9</i>  |
| <b>USING CATALYST.....</b>                                | <b>11</b> |
| <b>USING THE CATALYST LOADER.....</b>                     | <b>11</b> |
| <b>CATALYST FILE EXTENSIONS.....</b>                      | <b>11</b> |
| <b>THE SDCARD MEMORY MODE (SMM).....</b>                  | <b>12</b> |
| <b>CATALYST COMMANDS.....</b>                             | <b>13</b> |
| <i>Catalyst Internal Commands.....</i>                    | <i>13</i> |
| <b>DIR.....</b>   | <b>13</b> |
| <b>CLS.....</b>   | <b>13</b> |
| <b>CAT.....</b>   | <b>13</b> |
| <b>HELP.....</b>  | <b>13</b> |
| <i>Catalyst External Commands.....</i>                    | <i>13</i> |
| <b>LS.....</b>  | <b>13</b> |
| <b>MV.....</b>  | <b>14</b> |
| <b>RM.....</b>  | <b>14</b> |
| <b>CP.....</b>  | <b>15</b> |
| <b>MKDIR.....</b>   | <b>15</b> |
| <b>RMDIR.....</b>   | <b>15</b> |
| <i>Catalyst Optional Commands.....</i>                    | <i>17</i> |
| <b>BOOT n.....</b>  | <b>17</b> |
| <b>RESET n.....</b>                                       | <b>17</b> |
| <b>CATALYST AUTO-EXECUTION.....</b>                       | <b>17</b> |
| <b>CATALYST APPLICATIONS.....</b>                         | <b>18</b> |
| <b>DUMBO BASIC.....</b>                                   | <b>18</b> |
| <b>LUA.....</b>   | <b>18</b> |
| <b>JZIP.....</b>  | <b>19</b> |
| <b>PASCAL.....</b>  | <b>19</b> |
| <b>SUPER STAR TREK.....</b>                               | <b>20</b> |
| <b>VI.....</b>  | <b>20</b> |
| <b>PLATFORM-SPECIFIC NOTES.....</b>                       | <b>21</b> |
| <b>C3.....</b>  | <b>21</b> |
| <b>DRACBLADE.....</b>                                     | <b>21</b> |
| <b>HYDRA.....</b>   | <b>22</b> |
| <b>HYBRID.....</b>  | <b>22</b> |
| <b>MORPHEUS.....</b>                                      | <b>23</b> |
| <b>RAMBLADE.....</b>                                      | <b>26</b> |
| <b>TRIBLADEPROP.....</b>                                  | <b>26</b> |

[SUPERQUAD AND RAMPAGE.....](#) [27](#)  
[PROGRESS MESSAGES ON MULTI-PROP PLATFORMS.....](#) [27](#)  
**[CATALYST DEVELOPMENT.....](#) [28](#)**  
[REPORTING BUGS.....](#) [28](#)  
[IF YOU WANT TO HELP DEVELOP CATALYST.....](#) [28](#)  
[OKAY, BUT WHY IS IT CALLED "CATALYST".....](#) [28](#)  
[ACKNOWLEDGMENTS.....](#) [28](#)  
**[THE CURRENT CATALYST RELEASE.....](#) [29](#)**  
[WHAT'S NEW IN THIS RELEASE?.....](#) [29](#)  
[WHAT'S DUE IN THE NEXT RELEASE?.....](#) [29](#)

## What is Catalyst?

Catalina is an SD card-based program loader, plus a set of utility programs for the Parallax Propeller.

When used as intended, Catalyst looks very much like a fully functional Propeller operating system. However, strictly speaking, Catalyst is *just* a program loader - it is not really a true operating system because it does no resource management - however, it can be used to load programs that *do* perform various common resource management tasks – it even comes with a few - e.g. various utilities for doing SD card file management.

While it can be used with any Propeller programs, Catalyst is specifically intended to facilitate loading and using programs compiled with the Catalina C compiler.

It is not currently possible to compile Catalina C programs *on* the Propeller, but Catalyst comes with various other applications that can be used for self-hosted Propeller development, including the **vi** text editor, and such tools as a **BASIC** interpreter, a **Pascal** compiler and interpreter, and the **Lua** scripting language. It could also be used to edit, compile and then run SPIN programs using with the Sphinx SPIN compiler (not included – see <http://www.sphinxcompiler.com/>).

## Status

This is a full release of Catalyst. When you start the Catalyst loader, it will report itself as **Catalyst v3.3**. This version of Catalyst coincides with the release of Catalina 3.3.

## Features

- Compatible with any Propeller platform that supports Catalina and has an SD card available (e.g. Hydra, Hybrid, TriBladeProp, RamBlade, Morpheus, DracBlade, C3);
- Support for SPIN or Catalina LMM programs on *any* supported platform;
- Support for Catalina XMM programs on any supported platform with external XMM RAM (e.g. Hybrid, TriBladeProp, RamBlade, Morpheus, DracBlade, C3);
- Support for multi-CPU platforms (e.g. TriBladeProp, Morpheus);
- Support for the SDCARD Memory Mode, which maximizes the memory available to Catalina LMM programs on any supported platform;
- Provides familiar SD card file management (e.g. **ls**, **cp**, **mv**, **rm**, **mkdir**, **rmdir**);
- Supports self-hosted Propeller development (in Pascal, Basic and Lua).
- Supports passing command line parameters to both C and SPIN programs.

## **License**

All components of Catalyst are free and open source. Many of the components are licensed under the MIT license. Others are free but are licensed under the GNU General Public License, or other terms and conditions. All licenses are open-source, and free for non-commercial use – however, they are subject to various copyright and other conditions and you should consider the license terms of each component before using any of them in a commercial application.

## **Installing Catalyst**

### **Overview**

Catalyst is normally released as part of Catalina, and requires no further installation - but you must still build the appropriate binaries for your Propeller platform – see the section **Building Catalyst from source** (below).

If you received this as a separate release, there are two mandatory parts to Catalyst, and one optional part that may need to be installed:

From the binary release:

- The main Catalyst binary. This should be programmed into the EEPROM of the propeller. There will be a different version for each platform, and is distributed as part of the binary release for each specific propeller platform.
- The Catalyst external command and application program binaries. These programs should be loaded onto an SD card. They are distributed as part of a binary release for a specific propeller platform.

From the source release (optional):

- The Catalyst source code. This is distributed as part of the normal Catalina source release. It may also be distributed separately. In either case, it should be installed in the normal Catalina program directory (the default directory is *catalyst*)

### **Installing the Catalyst Binary Release**

#### **Load the Catalyst binary into EEPROM**

If you have downloaded a source release, you will first have to build Catalyst for your platform. Refer to the section **Building Catalyst from Source** (below).

If you have downloaded a binary release specifically for your Propeller platform, then you should be able to load Catalyst immediately. The main Catalyst binary is usually called **catalyst.binary** in the source release, but is called **catalyst.bin** in the binary release, to facilitate it being loaded onto an SD Card (which only supports 8.3 style file names). However, whether it is called **catalyst.binary** or **catalyst.bin**, it should be programmed into the EEPROM of the propeller. This can be done using the Catalina payload program if the Propeller is connected and turned on, using a command like:

```
payload -e catalyst.bin
```

### Load the Catalyst programs onto an SD Card

The remaining contents of the binary release must be copied an SD card (or micro SD card) that is then inserted into the Propeller.

This SD card must be formatted with a FAT file system. However, neither Catalyst nor Catalina support long file names, so all **.binary** file names must be renamed to be no longer than 8 characters, and all should have an extension of **.bin**.

Catalyst allows binary programs to be stored in a **/bin** directory on the SD card. If the program to be loaded exists in the root directory of the SD card then that version is loaded, otherwise Catalyst attempts to load the program from the **bin** directory (if it exists).

When copying the **.bin** binary files to an SD card, binary files can be either in the root directory, or within a **bin** directory. The rest of the files in the binary release should be copied to the root directory of the SD card.

## Installing the Catalyst Source Release

There is no installer for the Catalyst source release. If Catalyst is distributed separately from Catalina, simply use **unzip** (Windows version) or **gzip/tar**<sup>1</sup> (Linux version) to extract the source distribution into the folder in which Catalina is to be installed.

Under Windows, Catalyst should be installed in **C:\Program Files\Catalina\catalyst** and under Linux it should be installed in **\usr\local\lib\catalina\catalyst**.

Installing to a directory other than the default location is possible, but it means that some additional options will need to be specified (or some scripts modified) when compiling Catalyst.

Note that Catalyst 3.2 requires Catalina 3.2 – it should not be used with earlier versions of Catalina.

## Catalyst Directory Structure

When Catalyst is installed, the directory structure should be as follows:

```

Catalina
|
+--- catalyst
|   |
|   +--- bin
|   |
|   +--- core
|   |
|   +--- dumbo_basic
|   |
|   +--- jzip
|   |   |
|   |   +--- doc
|   |   |
|   +--- lua-5.1.4
|   |   |
|   |   +--- doc
|   |   +--- etc
|   |   +--- src
|   |   +--- test
|   |
|   +--- pascal
|   |   |
|   |   +--- p5_c
|   |   +--- p5_pascal
|   |   +--- ptoC
|   |
|   +--- demo
|   |
|   +--- sst
|   |
|   +--- xvi-2.47
|       |
|       +--- doc
|       +--- src

```

<sup>1</sup> Note that when using **tar**, the **-p** tar option should be specified to preserve file permissions.

## Building Catalyst from Source

The **catalyst** directory contains a **build\_all** script that can be used to build all platforms except **MORPHEUS** – for that platform there is a special **build\_morpheus** script. For example, some commands to build Catalyst would be:

```
build_all HYBRID
build_all TRIBLADEPROP CPU_2 PC VT100
build_morpheus
```

To build Catalyst under Windows you will require Catalina installed. If you also have MinGW and MSYS installed, then you can build all the programs using the **build\_all** script. The windows build scripts will attempt to detect if these are installed before proceeding (by trying to invoke the 'make' utility). Refer to the Catalina reference manual for more details on MinGW and MSYS.

If you do not have MinGW and MSYS installed, you should still build the core components using the **build\_all** script, but building some of the demo programs should then be done using Code::Blocks. There is a Code::Blocks workspace provided specifically designed to do this. To use it, follow these steps:

1. Start Code::Blocks (use the **Catalina Codeblocks** menu item in the **Catalina** program group in the Windows **Start** menu)
2. Open the Code::Blocks catalyst demos workspace file - the precise file name will depend on where you installed Catalina, but it will be called something similar to *C:\Program Files\Catalina\codeblocks\catalyst\_demos.workspace*.
3. Ensure that the option “*Explicitly add currently compiling file's directory to compiler search dirs*” option is selected on the **Other Settings** tab in the **Settings->Compiler and Debugger** dialog box – if this setting is not selected, you will get build errors.
4. Ensure that your Propeller platform has been correctly specified. Note that you can either do this *separately for each project* using the **Project->Build Options** command, or else *once for all projects* in the **Settings->Compiler and Debugger** command – but do not do it in both places!
5. Use the command **Build->Rebuild Workspace** to build all the catalyst demos. The binaries will automatically be placed in the **catalyst\bin** folder, along with the other Catalyst components previously built using the **build\_all** script.
6. Unfortunately, Code::Blocks generates each file with a **.binary** extension, whereas Catalyst expects an 8.3 character DOS style filename, so you must manually rename each binary generated using Code::Blocks (e.g. *vi.binary* should be renamed *vi.bin*).
7. Follow the directions given at the beginning of this section for installing the resulting Catalyst binaries.

See the **README.TXT** file in the **catalyst** directory for more details, and also the **README** files in each of the application directories.



## Compiling Programs for Catalyst

The main feature that Catalyst adds to existing programs is the ability to run them from the SD card, and to accept command line parameters to be passed to them on startup. This functionality is available both to Catalina C programs as well as SPIN programs.

### Compiling C programs to run under Catalyst

Nothing special is required to make C programs run under Catalyst. Any parameters specified on the Catalyst command line will be available to the C program in the normal **argc** and **argv** parameters.

An example program (*demo.c*) is contained in the *Catalyst/demo* directory. To compile it, use a command like:

```
catalina demo.c -lc -D HYBRID
```

For programs that do not need to accept command line parameters, the **NO\_ARGS** option can be specified during compilation – e.g.:

```
catalina demo.c -lc -D HYBRID -D NO_ARGS
```

Of course, in the case of the *demo.c* program, using this option makes the whole program a bit pointless since printing out its arguments is all the demo program does – but for some programs using this option can save a small amount of Hub RAM.

To compile a program to use the SDCARD Memory Mode (SMM), the executable must have a **.smm** extension, so use a command like:

```
catalina demo.c -lc -D HYBRID -D SDCARD -o demo.smm
```

The advantage of the SDCARD Memory Mode is that it uses a two-phase loader, with all the plugins and the kernel itself loaded separately from the C program. This means that on LMM platforms, a C program can use all the available Hub RAM (at least up to 31kb) as *application code* space, instead of being limited to what is left after all the plugins and the kernel itself are included (which can be up to 16kb, or *half the available Hub RAM!*). Without SMM mode, the space occupied by these is available as *data* space to the C program, but not as *code* space.

### Compiling SPIN programs to run under Catalyst

To enable SPIN programs to interpret command line parameters passed by Catalyst, a special SPIN module is provided called **Catalyst\_Arguments.spin**.

An example program (**demo.spin**) is contained in the *Catalyst/demo* directory. To compile it, use any SPIN compiler to produce a binary output. For example, to use homespun:

```
homespun demo.spin -b
```

Note that the example program must be manually modified to suit the platform on which you intend to run it – by default it is configured to run on a **HYBRID**. The clock speed, pins and perhaps the TV and video drivers will need to be manually modified to suit other platforms (as is normal for Spin programs).

The **Catalyst\_Arguments** module provides three methods:

- init(buffer)** *buffer* must point to a buffer of 1200 bytes (300 longs). This method must be called before any of the other argument methods.
- argc** this function returns the number of arguments (which may be zero).
- argv(i)** this function returns a pointer to a zero terminated string that contains the *i*th command line argument.

## Using Catalyst

### *Using the Catalyst Loader*

The main Catalyst binary is intended to be executed in interactive mode whenever the propeller is reset, to allow the entry of commands. Normally, this means the main Catalyst executable is loaded into EEPROM. However, Catalyst can also be used in a non-interactive mode as a stand-alone loader for Catalina programs in other Propeller operating systems - see the section titled Catalyst Auto-Execution later in this document for more details on this option.

When executed in interactive mode, Catalyst should display a simple banner line similar to the following:

```
Catalyst v3.0  
>
```

Where this prompt appears will differ depending on the Propeller platform. For example, on the **RAMBLADE** the Catalyst HMI uses a serial terminal emulator (e.g. on a PC), whereas on the **HYBRID** Catalyst will a local TV display and PS/2 keyboard. On other platforms, Catalyst may use either a serial terminal, or a local TV or VGA display and keyboard depending on the configuration parameters used when Catalyst is compiled. See the notes on each supported platform given later in this document.

When Catalyst is configured to use the PC serial terminal emulator HMI option, some of the commands expect a VT100 compatible terminal emulator. The recommend terminal emulator is **putty** - it is free, and versions for Windows are available from:

```
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
```

At this point, Catalyst commands can be entered. Catalyst contains a few simple *internal* commands (e.g. **dir**, **cat**, **help**) described below, but most of the Catalyst commands are *external* commands, loaded from the SD card.

Note that whenever the SD Card is removed and re-inserted, Catalyst must be reset. Otherwise the SD file system access will not work correctly.

### *Catalyst File Extensions*

Catalyst can be used to load and execute normal SPIN binaries as well as Catalina LMM binaries. Generally, such files should be placed on the SD Card with a **.bin** extension (the extension does not need to be specified when executing the program).

However, the main reason for using Catalyst is that it also knows how to load Catalina XMM programs, and Catalina SMM programs. Here are the rules Catalyst applies when loading and executing a file:

1. If a file extension is specified, it is used. Otherwise the loader looks for the file name using the following extensions (in the order shown), and it loads the first one it finds:

**.xmm**

**.smm**

**.lmm**

**.bin**

Catalyst will try two places when searching for each file name:

The *root* directory of the SD Card;

The **bin** directory of the SD Card (if one exists);

2. If the file has a **.xmm** extension, it is loaded as an XMM file;
3. If the file has a **.smm** extension, it is loaded as an SMM file;
4. If the file has *any other* extension, then it is loaded as an LMM file if it is 32kb or under in size, or an XMM file if it is over 32kb in size.

### ***The SDCARD Memory Mode (SMM)***

The SMM memory mode is a unique feature of Catalyst. It allows Catalina LMM programs to make much more effective use of the Propeller's limited Hub RAM by loading the program in two phases. This means all plugins/cog programs are loaded first, then the application program is loaded separately – allowing up to 31kb to be used for Catalina application program code. Compare this to normal programs (both Spin and Catalina LMM programs) where up to 16kb of Hub Ram may be needed simply to hold the programs to be loaded into the various cogs – which makes this space unavailable for use as application program code at run time (it can still be used as data space, but not as application code).

This means that using the SMM memory mode can almost **double** the amount of code space available to Catalina LMM programs!

The SDCARD memory model is enabled by defining the **SDCARD** symbol on the command line, and also by saving the program with a **.smm** extension (instead of a **.bin** extension). For example:

```
catalina othello.c -lc -D SDCARD
mv othello.binary othello.smm
```

A more detailed description of the SMM load process is given in the file *README.SMM\_Loader* in the main Catalina directory.

There are some limitations when using the SMM memory model:

- SMM programs must be 31kb or less (less if they use plugins that allocate buffer space in Hub RAM).
- SMM programs will load the SD plugin even if the program would not normally need it. This may mean the program requires one more cog than usual.

## **Catalyst Commands**

### **Catalyst Internal Commands**

The internal commands are built into the Catalyst loader. They will work on any Propeller platform supported by Catalina.

Note that when using a local display and keyboard, all the internal commands will pause after each screen full of information has been displayed, and require you to press a key to continue. This is not the case when using the PC terminal HMI, since Catalyst assumes a terminal emulator is used which can be scrolled back to see any information that scrolls off the screen.

#### **DIR**

Display the files in the SD card root directory. This command does not accept any parameters, so only files in the root directory are displayed. For a more comprehensive display, see the LS command.

#### **CLS**

Clear the screen.

For CLS to work correctly when using a PC HMI, a VT100 compatible terminal emulator (such as putty) must be used.

#### **CAT**

Display a text file. On platforms that use a local display, the file is displayed one page at a time. The ESC key can be used to exit at each pause.

#### **HELP**

Display some simple help about the various internal commands.

### **Catalyst External Commands**

The external commands may be normal SPIN programs or LMM C programs. They will work on any Propeller platform supported by Catalina. They do not require XMM RAM, but they depend on being loaded by Catalyst so that command line arguments can be entered.

#### **LS**

list the details of a file or the contents of a directory.

syntax:

```
ls [options] [directory]
```

options:

```
-h or -?    print help  
-l         long format listing  
-r         recursively list subdirectories
```

e.g. to list the current (top level) directory:

```
ls
```

or

```
ls .
```

To list directory a/b/c:

```
ls a/b/c
```

## **MV**

Move one file to another, or one or more files to a directory. If there are only two arguments, and the target does not exist, you must tell mv whether the target is supposed to be a file or a directory.

NOTE: mv is essentially a cp followed by an rm - with the rm only performed if the copy succeeds. This means that there must be enough free space to hold two complete copies of the file.

syntax:

```
mv [options] src_file [src_file ...] target_file_or_directory
```

options:

```
-h or -?    print help
-f          force overwrite (if target is read-only)
-I          interactive (prompt for each move)
-t          target is a directory
-T          target is a file
```

e.g:

```
mv a.txt b.txt c.txt my_dir
mv a.txt b.txt
```

## **RM**

Remove one or more files, optionally also removing directories (provided they are empty).

syntax:

```
rm [options] file_or_directory ...
```

options:

```
-h or -?    print help
-f          remove empty directories
```

e.g:

```
rm a/b.txt
rm a.txt b.txt
```

**CP**

Copy one file to another, or one or more files to a directory. If there are only two arguments, and the target does not exist, you must tell cp whether the target is supposed to be a file or a directory.

syntax:

```
cp [options] src_file [src_file ...] target_file_or_directory
```

options:

```
-h or -?    print help
-f          force overwrite (if target is read-only)
-I         interactive (prompt for each copy)
-t         target is a directory
-T         target is a file
```

e.g:

```
cp a.txt b.txt c.txt my_dir
cp a.txt b.txt
```

**MKDIR**

Make one or more directories, optionally making each parent directory in turn if they do not exist.

syntax:

```
mkdir [options] directory ...
```

options:

```
-h or -?    print help
-p          create parent directories if required
```

e.g:

```
mkdir a/b/c    <- will make directory c only if a/b exists
mkdir -p a/b/c <- will make directories a, then a/b,
                then a/b/c if they do not already exist
```

**RMDIR**

Remove one or more directories, optionally removing each parent directory recursively (if they are empty).

syntax:

```
rmdir [options] directory ...
```

options:

```
-h or -?    print help
-p          remove parent directories if empty
```

e.g. to remove directory c (but leave a and b intact):

```
rm -r a/b/c
```

To remove directory a/b/c, then a/b, then a (provided they are empty):

```
rm -p a/b/c
```



## Catalyst Optional Commands

On platforms that contain multiple CPUs (such as the **TRIBLADEPROP**, or **MORPHEUS**) it may also be convenient to have the normal Catalina multi-CPU utilities loaded onto the Catalyst SD card.

Note that the relevant Catalina utilities are called **CPU\_n\_Boot.spin** and **CPU\_n\_Reset.spin**, and when compiled they may be called something like **BOOTn.m** (e.g. **BOOT2.1**) - but for Catalyst they should be renamed to simply **BOOT\_n.bin** or **RESET\_n.bin** when the compiled binaries are copied to the SD card (this is not absolutely required, but convention Catalyst uses **.bin** as an executable file name extension).

### **BOOT\_n**

Reboot the specified Propeller, loading the Catalina Generic SIO Binary Loader so that another program can be loaded.

syntax:

```
boot_1          (TRIBLADEPROP only)
boot_2          (MORPHEUS only)
boot_3          (TRIBLADEPROP only)
```

### **RESET\_n**

Just reset the specified Propeller. Whatever program is loaded into EEPROM will be run.

syntax:

```
reset_1        (TRIBLADEPROP only)
reset_2        (MORPHEUS only)
reset_3        (TRIBLADEPROP only)
```

## Catalyst Auto-Execution

To facilitate the use of Catalyst as a loader from within other operating systems, Catalyst will check on startup for the existence of the file **AUTOEXEC.TXT** in the root directory of the SD Card. If it finds this file, it will read one command out of the file (up to the first zero byte, line terminator or EOF) and attempt to execute it as a Catalyst command – including parameters. If the command execution fails, Catalyst will enter interactive mode, otherwise the specified command (which may be any SPIN or C program executable) will be run, with the specified command line parameters. For example, the file may contain a command such as:

```
vi autoexec.txt
```

This would start the vi editor on the **AUTOEXEC.TXT** file itself!

This facility allows other Propeller operating systems to load Catalina LMM, SMM, or XMM programs without having to include the appropriate Catalina loader – Catalyst already incorporates the code to load such programs, and is used strictly as a non-interactive program loader. All the host operating system has to do is create an appropriate **AUTOEXEC.TXT** file and execute **Catalyst.binary** – which is a normal Propeller executable.

When compiling Catalyst, you can optionally tell it to delete the **AUTOEXEC.TXT** file after it has been executed once. To do this, define the symbol **AUTODELETE** when building Catalyst. For example:

```
build_all HYBRID AUTODELETE
```

This might be appropriate if you mainly intended to use Catalyst as an XMM program loader from within another operating system. It would *not* be appropriate if you were using Catalyst to auto execute an XMM application in an embedded environment.

## ***Catalyst Applications***

Catalyst provides a rich set of application programs. However, all the example applications provided require XMM RAM to run. Some will run only on platforms with 512k of XMM RAM or more installed.

This section does not describe each application in detail – it only describes how to run the application programs from the Catalyst command line. See the individual application program documentation for more details on the application itself.

### ***DUMBO BASIC***

Load the Dumbo BASIC interpreter. Note that Dumbo BASIC is just an interpreter – the programs must be created externally (e.g. using the vi text editor).

syntax:

```
dbasic [ basic_program.bas ]
```

e.g:

```
dbasic eliza.bas
```

If no parameter is specified, dbasic will prompt for the name of the basic file to execute.

### ***LUA***

Load the Lua interpreter (**lua**), or run the Lua compiler (**luac**).

syntax:

```
lua [script.lua]
```

```
luac -o output_filename script.lua
```

e.g:

```
lua fact.lua
```

```
luac -o f.lua fact.lua
```

If no file is specified to the **lua** command, commands can be entered directly on the terminal.

The Lua compiler (**luac**) compiles a Lua program to byte code, which speeds up loading – but the resulting file must still be executed with **lua**.

NOTE: when using **luac**, do not omit the **-o** parameter, or **lua** will output the binary result to the terminal.

## JZIP

Load the JZIP Infocom game interpreter. The number of rows and columns to use for the screen size can be specified on the command line, but the game will detect the actual screen size when local devices are used, and assume 80x24 when using the PC HMI option.

syntax:

```
jzip [-ccols] [-lrows] [ game.dat ]
```

e.g:

```
jzip zork1.dat
```

If no parameter is specified, jzip will prompt for the name of the game file to execute.

## PASCAL

Load the Pascal interpreter (**pint**), or run the Pascal compiler (**pcom**).

syntax:

```
pint [compiled_program]
```

```
pcom [ program_to_compile [compiled_program] ]
```

e.g:

```
pint startrek.p5
```

```
pcom startrek.pas startrek.p5
```

The output files from the compiler must be executed with the interpreter.

If no file is specified to the **pcom** or **pint** commands, the programs will prompt for a file name.

By convention, the compiled version of the Pascal program **prog.pas** is normally called **prog.p5**

In addition to a few sample programs, two precompiled programs are provided – **startrek.p5** and **basics.p5**. The first is yet another version of the classic Start Trek game and the second is a basic interpreter. These programs are provided compiled because they can each take several hours to compile on the Propeller – even loading the precompiled programs can take a minute or two.

**SUPER STAR TREK**

Play a game of Super Star Trek.

syntax:

```
sst
```

There are no parameters to this command. See the document sst.doc for help.

**VI**

Load the XVI text editor (the binary is renamed to vi for convenience). The program accepts various options – see the xvi documentation for more details.

syntax:

```
vi [options] [ filename ... ]
```

e.g:

```
vi sample1.txt
```

A common option to specify is **-s format=msdos** or **-s format=unix** to specify whether msdos or unix line termination is to be used (the default is unis, so if you open an msdos file you may see extraneous **^M** characters at the end of each line).

If more than one filename is specified, vi will open the first two in separate windows. After that, use **:n** (i.e. *colon n*) to move to the next file.

## Platform-specific Notes

### C3

All the binaries in this release (as well as Catalyst itself) are built to use a High resolution NTSC TV and keyboard connected to the Propeller.

Note: After each external command or demo program is run, the screen is cleared. Catalyst will usually ask you to enter a key to continue so that you can read the output of the command.

On the C3, the SPI Flash is used to execute the external demo programs. This means loading programs can take several seconds, during which time there may be no indication that the command is being processed.

Also, the C3 has only 64kb of SPI Ram available, which is not sufficient space to execute some of the larger demo programs – notably the pcom/pint pascal interpreter, or the jzip interpreter.

Here are some example commands you could try:

```

cat SAMPLE.PAS           <---- list a text file
vi CATALYST.TXT         <---- edit a text file
dbasic STARTREK.BAS     <---- run a basic program
mkdir my_dir            <---- make a directory
vi my_dir/my_file.txt   <---- edit a file in a directory
ls my_dir               <---- list the contents of a directory
rm my_dir/my_file.txt   <---- remove a file from a directory
sst                     <---- play a game of Super Star Trek
dbasic ELIZA.BAS        <---- get some psychiatric help

```

On the C3, it is possible to recompile Catalyst to use a VGA or PC HMI option if desired.

### DracBlade

On the DracBlade, all the Catalyst binaries are built to use a High resolution VGA HMI plugin, which uses the display and keyboard connected to the Propeller.

Note: After each external command or demo program is run, the screen is cleared. Catalyst will usually ask you to enter a key to continue so that you can read the output of the command.

Here are some example commands you could try:

```

cat SAMPLE.PAS           <---- list a text file
pcom SAMPLE.PAS SAMPLE.P5 <---- compile a pascal program
pint SAMPLE.P%          <---- run the compiled program
vi CATALYST.TXT         <---- edit a text file

```

```

dbasic STARTREK.BAS      <---- run a basic program
mkdir my_dir             <---- make a directory
vi my_dir/my_file.txt    <---- edit a file in a directory
ls my_dir                <---- list the contents of a directory
rm my_dir/my_file.txt    <---- remove a file from a directory
jzip ZORK3.DAT           <---- play a game of Zork
sst                      <---- play a game of Super Star Trek
dbasic ELIZA.BAS         <---- get some psychiatric help

```

On the DracBlade, it is possible to recompile Catalyst (or some parts of Catalyst) to use a PC HMI option (or a low resolution VGA option). This may be required to run some large applications (such as the Pascal compiler) since the High Resolution VGA driver consumes a large amount of Hub RAM space, which limits the stack space available to other programs.

## Hydra

All the binaries in this release (as well as Catalyst itself) are built to use a High resolution NTSC TV and keyboard connected to the Propeller.

Note: After each external command or demo program is run, the screen is cleared. Catalyst will usually ask you to enter a key to continue so that you can read the output of the command.

The Hydra cannot simultaneously use the SD Card and XMM RAM, so while Catalyst itself runs, and the demo programs can be compiled and loaded serially (using Payload) none of the demo programs can be loaded from SD Card using Catalyst. However, both Catalina LMM programs and normal Spin programs can be loaded and run with Catalyst.

Here are some example commands you could try:

```

cat SAMPLE.PAS          <---- list a text file
vi CATALYST.TXT         <---- edit a text file
mkdir my_dir            <---- make a directory
ls my_dir               <---- list the contents of a directory
rm my_dir/my_file.txt   <---- remove a file from a directory

```

On the Hydra, it is **NOT** possible to recompile Catalyst to use a PC HMI option if the XMM RAM is being used – the HX512 does not allow the serial port to be used at the same time.

## Hybrid

All the binaries in this release (as well as Catalyst itself) are built to use a High resolution NTSC TV and keyboard connected to the Propeller.

Note: After each external command or demo program is run, the screen is cleared. Catalyst will usually ask you to enter a key to continue so that you can read the output of the command.

Here are some example commands you could try:

```

cat SAMPLE.PAS          <---- list a text file
pcom SAMPLE.PAS SAMPLE.P5 <---- compile a pascal program
pint SAMPLE.P%         <---- run the compiled program
vi CATALYST.TXT       <---- edit a text file
dbasic STARTREK.BAS   <---- run a basic program
mkdir my_dir          <---- make a directory
vi my_dir/my_file.txt <---- edit a file in a directory
ls my_dir             <---- list the contents of a directory
rm my_dir/my_file.txt <---- remove a file from a directory
jzip ZORK3.DAT        <---- play a game of Zork
sst                   <---- play a game of Super Star Trek
dbasic ELIZA.BAS      <---- get some psychiatric help

```

On the Hybrid, it is **NOT** possible to recompile Catalyst to use a PC HMI option if the XMM RAM is being used – the HX512 does not allow the serial port to be used at the same time.

## **Morpheus**

On Morpheus, Catalyst is configured to use the PC HMI option, and some programs expect a VT100 compatible PC Terminal emulator (such as **putty**).

On Morpheus, running the external commands often results in rubbish being displayed on the screen at the end of each command - this is because each external command resets the Propeller at the end. This means you may need to scroll the putty screen up to see the output of the command. While this is a little annoying, it does not really affect the operation of Catalyst.

What *does* makes thing complex on Morpheus is that Catalyst runs on CPU #1, but the demo programs must be run on CPU #2 since they all require access to XMM RAM. They must use Catalina proxy drivers since they need access to both the XMM RAM and the SD Card.

The binary distribution of Catalyst also includes the normal Catalina multi-cpu utilities for Morpheus (renamed to be a bit more user-friendly):

```

BOOT_2.BIN - start a boot loader program running on CPU #2
RESET_2.BIN - reset CPU #2
LOAD_2.BIN - SIO Loader (can be programmed into EEPROM on CPU #2)

```

The following explains how to run each Catalyst demo program:

**vi** - the client executable runs on CPU\_2, using the SD card on CPU\_1. This can make editing a bit slow on large files! To run it, you need to load both the client and the server. If you have not already done so, you also first need to load the boot loader into CPU #2:

```

boot_2          <- load the boot loader into CPU #2
@              <- to select CPU #2 for next load
vi             <- load the client into CPU #2 (note that this
               may take a while!)
proxy          <- load the server into CPU #1

```

There are a few things to note about loading/running this program:

1. If you program the Generic SIO program loader (**load\_2.bin**) into the eeprom of CPU #2, you do not need to enter any of the **boot\_2** commands – loading the SIO loader into CPU #2 is really all the **boot\_2** program does.
2. If you have a VGA display connected to CPU 2, you will see progress messages displayed on the screen as the client program is loaded.
3. You may see rubbish on the screen when the client program starts up (and before you get a chance to start the proxy server) – this occurs when the program loader restarts the Propeller to run the loaded program. In some cases, these rubbish characters can cause the terminal emulator program (e.g. putty) to lock up – just restart the terminal emulator and everything should be ok.
4. You cannot currently use command line parameters for programs loaded into another CPU - e.g. you cannot pass parameters to the **vi** client. Instead, to edit a particular file, once vi starts you can enter the command:

```
:e filename
```

5. Just because you quit a demo client program (i.e. **vi**) does not mean you will return to the Catalyst prompt. This is because the proxy program is still running. Since there is currently no way to tell the proxy program to terminate, to return to the Catalyst prompt you must manually reset the Propeller.
6. If you accidentally try to run the **vi** client on CPU #1, you will see a series of **ÿÿÿÿ** characters - this is the client program polling for the proxy server. You will have to manually reset the prop.

**sst** -you must run the client on CPU #2, and the proxy server on CPU #1:

```

boot_2          <- load the boot loader into CPU #2
@              <- to select CPU #2 for next load
sst            <- load the client into CPU #2
proxy          <- load the server into CPU #1

```



The same provisos apply as above - i.e. you cannot enter command line arguments to the client, and you must reboot the Propeller to return to the Catalyst prompt.

**jzip** - you must run the client on CPU #2, and the proxy server on CPU #1:

```
boot_2      <- load the boot loader into CPU #2
@           <- to select CPU #2 for next load
jzip        <- load the client into CPU #2
proxy       <- load the proxy server into CPU #1
```

The same provisos apply as above - i.e. you cannot enter command line arguments to the client, and you must reboot the Propeller to return to the Catalyst prompt. The program will prompt you for a game file to run.

**dbasic** - you must run the client on CPU #2, and the proxy server on CPU #1:

```
boot_2      <- load the boot loader into CPU #2
@           <- to select CPU #2 for next load
dbasic       <- load the client into CPU #2
proxy        <- load the proxy server into CPU #1
```

The same provisos apply as above - i.e. you cannot enter command line arguments to the client, and you must reboot the Propeller to return to the Catalyst prompt. The program will prompt you for a basic file to run.

**pcom/pint** - you must run the clients on CPU#2, and the proxy server on CPU #1:

```
boot_2      <- load the boot loader into CPU #2
@           <- to select CPU #2 for next load
pcom         <- load the compiler client into CPU #2
proxy        <- load the proxy server into CPU #1
```

or

```
boot_2      <- load the boot loader into CPU #2
@           <- to select CPU #2 for next load
pint         <- load the interpreter client into CPU #2
proxy        <- load the proxy server into CPU #1
```

The same provisos apply as above - i.e. you cannot enter command line arguments to the client, and you must reboot the Propeller to return to the Catalyst prompt. The program will prompt you for a Pascal file to compiler or interpret.

**lua** -you must run the client on CPU #2, and the proxy server on CPU #1:

```
boot_2      <- load the boot loader into CPU #2
@           <- to select CPU #2 for next load
lua         <- load the client into CPU #2
proxy       <- load the server into CPU #1
```

The same provisos apply as above - i.e. you cannot enter command line arguments to the client, and you must reboot the Propeller to return to the Catalyst prompt. Instead of command line parameters, you must use the Lua **dofile** command to get lua to execute a script (e.g. enter **dofile("hello.lua")**) - see the Lua documentation, or the **README.Lua** file for more details.

### ***RamBlade***

On the RamBlade, Catalyst is configured to use the PC HMI option, and some programs expect a VT100 compatible PC Terminal emulator (such as **putty**).

Here are some example commands you could try:

```

cat SAMPLE.PAS           <---- list a text file
pcom SAMPLE.PAS SAMPLE.P5 <---- compile a pascal program
pint SAMPLE.P%          <---- run the compiled program
vi CATALYST.TXT         <---- edit a text file
dbasic STARTREK.BAS     <---- run a basic program
mkdir my_dir            <---- make a directory
vi my_dir/my_file.txt   <---- edit a file in a directory
ls my_dir               <---- list the contents of a directory
rm my_dir/my_file.txt   <---- remove a file from a directory
jzip ZORK3.DAT          <---- play a game of Zork
sst                     <---- play a game of Super Star Trek
dbasic ELIZA.BAS        <---- get some psychiatric help

```

### ***TriBladeProp***

On the TriBladeProp, Catalyst is configured to use the PC HMI option, and some programs expect a VT100 compatible PC Terminal emulator (such as **putty**).

Here are some example commands you could try:

```

cat SAMPLE.PAS           <---- list a text file
pcom SAMPLE.PAS SAMPLE.P5 <---- compile a pascal program
pint SAMPLE.P%          <---- run the compiled program
vi CATALYST.TXT         <---- edit a text file
dbasic STARTREK.BAS     <---- run a basic program
mkdir my_dir            <---- make a directory
vi my_dir/my_file.txt   <---- edit a file in a directory
ls my_dir               <---- list the contents of a directory
rm my_dir/my_file.txt   <---- remove a file from a directory
jzip ZORK3.DAT          <---- play a game of Zork
sst                     <---- play a game of Super Star Trek

```

```
dbasic ELIZA.BAS          <---- get some psychiatric help
```

It would be possible to recompile Catalyst to use the local display and keyboard on CPU #1, via a proxy driver from CPU #2 (which has access to the SD card). The various **build\_morpheus** scripts show this can be accomplished using proxy devices.

### ***SuperQuad and RamPage***

The SuperQuad and RamPage are not separate platforms – refer to the notes appropriate to the platform to which they are attached.

However, note that the SuperQuad has no XMM RAM, so only SMALL mode programs can be created. This mode does not provide sufficient RAM space to execute any of the larger demo programs.

### ***Progress Messages on Multi-Prop platforms***

On multi-prop platforms (such as the TriBladeProp and Morpheus), it is routine to have to load programs into different CPUs. On platforms where the target CPU supports a directly connected screen, Catalyst will display progress messages on that screen (e.g. on a TV connected to the TriBladeProp CPU 1, or a VGA display connected to the Morpheus CPU 2).

This makes it easy to tell that the program is in fact being loaded correctly - but displaying this information can both slow down the load process slightly, and also reduce the size of programs that can be loaded.

Therefore, this functionality can be disabled. It is enabled by defining the **DISPLAY\_LOAD** command line symbol when compiling the programs in the *Catalina/utilities* folder.

To disable this functionality, edit the **build\_all** script (**build\_all.bat** under Windows) in the *Catalina/utilities* folder and look for **-D DISPLAY\_LOAD**. Simply delete this wherever it occurs (including the **-D**). Then recompile Catalyst.

## Catalyst Development

### **Reporting Bugs**

Please report all Catalyst bugs to [ross@thevastydeep.com](mailto:ross@thevastydeep.com).

Where possible, please include a *brief* example that demonstrates the problem.

### **If you want to help develop Catalyst**

Anyone who has ideas or wants to assist in the development of Catalyst should contact Ross Higson at [ross@thevastydeep.com](mailto:ross@thevastydeep.com)

### **Okay, but why is it called “Catalyst”?**

In chemistry a catalyst is a substance that facilitates a chemical reaction, but is not itself consumed. Catalyst is intended to facilitate the use of Catalina on the Propeller, but it does not itself consume any Propeller resources.

### **Acknowledgments**

Kye, for his FATEngine module.

Dr\_Acula, for his work on the auto-execute mode.

## The Current Catalyst Release

### *What's new in this release?*

- Add SUPERQUAD and RAMPAGE support.

### *What's due in the next release?*

- Add wildcard support to ls, cp, mv and rm
- Add 'flush' so that we don't have to wait for a fixed duration after each command when using the PC HMI option.
- Add a way to exit from the proxy program, so we don't always have to reset the propeller after each proxied command.