Catalina Release History

Introduction

The release history was getting too large to keep including it in the Catalina reference manual, so it has now been moved to its own document.

This release history is mainly of interest to maintainers of Catalina, or those users who have installed a previous version of Catalina and who wish to understand the benefits or consequences of upgrading their version.

It is of little or no interest to new Catalina users, who should read the **Catalina Reference Manual** or the various tutorial documents instead.

Release 8.5

New Functionality

1. Catalina now supports WiFi-based Remote Procedure Calls (RPCs) in addition to ALOHA serial RPCs. These are now implemented alongside the ALOHA protocol in several eLua variants. It is possible to have just ALOHA services, just WiFi services, or a combination of both. All the existing ALOHA example programs have been updated to use either ALOHA or WiFi, and an example of using both in the same program is given in a new 'hybrid' example. See the document **ALOHA from Lua** for more details.

2. The binaries generated by compiling the ALOHA versions of eLua have been renamed **alua**, **aluax** etc to be consistent with the new WiFi RPC versions of eLua (which are called **rlua**, **rluax** etc). No changes to functionality.

3. The eLua programs now load a generic 'serial' module if Lua is compiled with either the 2 port or 8 port serial plugin library (i.e. **-lserial2** or **-lserial8**). Lua programs can use the 'serial' module without needing to know if it is using the 2 port serial or 8 port serial plugin.

4. Lua now includes a module ('wifi') that allows access to the C WiFi support functions. The module is loaded automatically if Lua is linked with the 'wifi' library (e.g. compiled with **-lwifi**).

5. All the pre-compiled versions of the eLua demos have been removed, since they will all need to be modified and recompiled to use the new WiFi RPC capabilities.

6. Base64 encode and decode routines have been added to the Catalina library. See *include/base64.h* for details.

7. The buffer size of the 8 port serial plugin has been increased from 32 bytes to 1024 bytes to accommodate the new WiFi RPC functionality. Since there are 16 such buffers (one for each direction of each port) this can make programs that use this plugin up to 16k larger. If this causes problems, the files target\p2\s8serial.t and source\lib\serial8\core.c both need to be modified, and the Catalina library must be recompiled.

8. The WiFi definitions have been removed from the **P2_MASTER** and **P2_SLAVE** platform configuration files (i.e. *P2MASTER.inc* and *P2SLAVE.inc*) since this interfered with the ALOHA serial functionality. Instead, two new platform configurations have been added -

P2_WIFI_MASTER and **P2_WIFI_SLAVE** (i.e. *P2WIFI_M.inc* and *P2_WIFI_S.inc*) which can be used if both WiFi RPC and ALOHA serial RPC capabilities are required. If only the WiFi RPC capabilities are required, **P2_WIFI** can still be used.

9. Catalyst now includes a new Lua script (*script.lua*) that can be used to invoke any Catalyst script from the command line just by typing the script name, without having to say **exec script**. See *script.lua* for more details.

Other Changes

1. A significant bug in the implementation of **getenv()** has been fixed. The bug affected only Propeller 2 programs that used **getenv()**, and Lua programs (because Lua uses **getenv()** during initialization), and it also generally only affected XMM programs. The bug caused memory corruption, usually on startup. Programs that started ok generally ran properly thereafter. A previous workaround to this bug was to add calls to **_align_sbrk()** to move the C heap to another location - this generally worked around the problem but didn't solve it. These calls have now been removed, but the **_align_sbrk()** function is still available for other purposes. Affected the Propeller 2 only.

Release 8.4

New Functionality

1. Support has been added for the Parallax ESP8266 WiFi module. It will be supported on both the Propeller 1 and 2, but it has only currently been tested on the Propeller 2. A new library (*libwifi*) has been added, which can be used by adding **-lwifi** to the Catalina command. Serial communication with the WiFi module must be done using the 2 or 8 port serial comms module (Propeller 2) or the 4 port serial comms module (Propeller 1).

   To simplify using the WiFi module on the Propeller 2, a new platform configuration file has been added (*P2WIFI.inc*) which is enabled by defining the Catalina symbol **P2_WIFI**. It is generally the same as *P2EDGE.inc* except that it assumes a WiFi module is installed on pins 16 .. 23. So for the Propeller 2 you might use a command like:

   ```
   catalina -p2 testwifi.c -lc -lwifi -lserial2 -C P2_WIFI
   ```

   On the Propeller 1, the serial4 configuration must be specified in the file *target/p1/Extras.spin* but there is no specific configuration of the WiFI module in the Propeller 1 target files - instead, the pins etc must be specified in the application program when the WiFi module is initialized.

   So (for example) on the Propeller 1 Activity board, you might use a Catalina command like:

   ```
   catalina testwifi.c -lc -lwifi -lserial4 -C ACTIVITY
   ```

   Note that on the Propeller 1, if a WiFi program is too large to fit in the Propeller 1 Hub RAM, you may need to add additional options to optimize the program (**-O5**), use COMPACT mode (**-C COMPACT**), or use the EEPROM load option (**-C EEPROM**).

See the include file *include/wifi.h* and the example programs in the *demos/wifi* folder, and the *README.TXT* file in that folder for more details.

2. To accommodate the use of different serial plugins with the WiFi module, the 2, 4 and 8 port serial libraries now all use the same names for their functions - i.e. names like **s_xxxx()** instead of **s2_xxxx()**, **s4_xxxx()** and **s8_xxxx()** (respectively). The previous names can still be used by including the relevant serial header file (e.g. *include/serial2.h* which now include #defines like defining **s2_tx()** to be **s_tx()**) or the new general-purpose header file *include/serial.h* which detects which serial plugin and library is in use and includes the relevant serial header file.

3. On the Propeller 2, the default HMI option is now the **SIMPLE** serial interface, rather than **TTY**. The reason for this change is that on the Propeller 2, the TTY HMI actually uses one port of the 2 port serial plugin, which means that the default HMI option would no longer work with the WiFi module and the serial2 plugin (it would lead to errors about multiply defined symbols). To reinstate the previous behaviour, just explicitly add **-C TTY** to the catalina command.

   Because a serial comms plugin must be loaded to communicate with the WiFi module, and all the serial comms plugins support at least two ports, the WiFi programs typically use the first serial port (port 0) to do that, and the second serial port (port 1) for user interactions. Therefore, it is common for WiFi programs to not load any HMI plugins at all (specifying **-C NO_HMI**) and not use any stdio functions at all - this saves both Hub RAM and cogs.

Other Changes

2. The serial header files (*tty.h*, *serial2.h*, *serial4.h* and *serial8.h*) had some errors which may have resulted in syntax errors during a compilation if some of the definitions were used. Also, using the **cls** functions (e.g. **s2_cls()** may have resulted in an error about an undefined symbol (in this case **s2_char()**).

3. The tiny I/O library had a bug in the **scanf** and **sscanf** functions - if the last specifier in the format string was **%s** and the string being scanned did not have a white space at the end of it (and in particular, if the string being scanned was terminated by a null) then the scanning did not stop at the end of the string, and scanf and **sscanf** typically returned rubbish and may have corrupted memory.

Release 8.3

New Functionality

1. Payload now accepts multiple **-q** options, with all the **-q** options being 'or'ed together to determine the final modes. So, for example, specifying **-q1 -q2** is now the same as specifying **-q3**.

2. Payload now accepts an additional **-q** option (**-q32**) which specifies that payload should interpret CR and LF and move the screen cursor even if these keys are otherwise being ignored. This means that payload handles the screen movements rather than the ncurses package on Linux (or the

pdcurses package on Windows). This can solve some compatibility issues with C programs that expect LF to behave like a C style 'new line' but also expect the ENTER key to generate a 'carriage return' rather than a 'line feed' (this is something that ncurses/pdcurses cannot do).

3. Some payload options have changed descriptions to be more correct, and also to correspond more closely with their Comms program equivalents. For instance, the description of the payload -q16 option has changed from "CR to LF on Output" to "Auto CR on LF Output", which is the same as used by the Comms program (note that the functionality has not changed - the description in payload was incorrect).

4. A CR/LF test program (called *crlftest.c*) has been added to the demos folder. It can be used to test the effects of various payload and Comms options, and also various Catalina symbols that affect CR and LF processing.

5. It is now possible to build Catalyst and Lua without the 'linenoise' library, which was added by default when building Catalyst (for the Propeller 2), and Lua for the Propeller 1 or 2). The 'linenoise' library can now be disabled by defining the Catalina symbol **NO_LINENOISE**.

   Using the 'linenoise' library works ok with both payload internal and external terminal emulators, but not with the Parallax Serial Terminal.

6. None of the demo programs are now built by default with the Catalina symbol **CR_ON_LF** defined. This was originally done to make programs such as Catalyst compatible with the Parallax Serial Terminal, but since the Parallax Serial Terminal has issues with the 'linenoise' library (which adds command line editing and command history to both Catalyst and Lua), it now needs a special build anyway. So the defaults of both payload and Comms have been unified to eliminate the need to be compatible with the **CR_ON_LF** option, which eliminates the need to use the payload **-q** option or adjust the "Advanced Options" in the Comms program.

   The Parallax Serial Terminal can still be used if programs are compiled with the Catalins symbols **CR_ON_LF** and **NO_LINENOISE** defined.

7. Since the Propeller 2 Evaluation board is no longer readily available, the Propeller 2 **CUSTOM** definitions have been modified to use the **P2_EDGE** definitions rather than the **P2_EVAL** definitions. Also, the demo programs in *P2_DEMO.ZIP* are now built for the **P2_EDGE** rather than the **P2_EVAL**.

8. Separate instructions for building Catalina from source under the Raspberry Pi OS have been added to *BUILD.TXT*. These instructions will also work on other Debian-based Linux releases.

9. The self-hosted version of Catalina now includes the **spinc** utility. Since the output of this utility used to be sent to *stdout* and would have to be redirected to a file (which is not possible under Catalyst) a new command line option has also been added: **-g file** will now generate a file with with the output instead of sending it to *stdout*.

Other Changes

4. Defining the symbol **P2** either on the command line or in **CATALINA_DEFINE** incorrectly forced **NATIVE** mode, overriding any other specified mode (e.g. **TINY**, **CUSTOM**, **SMALL**, **LARGE**). Affected the Propeller 2 only on Windows, Linux, Pi OS and Catalyst.

5. The payload **-q4** and **-q8** options were being incorrectly applied to the output (to the screen) rather than the input (from the keyboard). Affected the Propeller 1 and 2 on Windows, Linux and Pi OS.

6. Various Linux scripts have been modified to remove the bash '-login' option, which made these scripts not work in the Pi OS or Debian-based Linux distributions. Affected the Propeller 1 and 2 on Linux and Pi OS.

7. The document **Getting Started with Catalina** was very out of date, and has been extensively revised and updated.

8. The Catalina Reference Manual (Propeller 1 and Propeller 2) said that the **CR_ON_LF** option translated CR to CR LF, but what it actually does is translate LF to CR LF.

9. The payload **-o** option now always sets the default baud rate as appropriate for the Propeller 1 (115200) or Propeller 2 (230400) unless a baud rate is explicitly set. Previously, it would only do so if a file to load was specified, not if only interactive mode was specified.

Release 8.2

Release 8.2 was never formally released. All changes were incorporated into release 8.3

Release 8.1.2

New Functionality

1. The default garbage collection for all Lua programs (including Lua itself, iLua, mLua and eLua) now has a "pause" value of 105, which means it will start a new cycle when the use of memory exceeds 105% of the use after the previous collection. This makes the garbage collector even more aggressive, which reduces the allocation of dynamic memory at the cost of some performance.

2. eLua now has two additional variants that include ALOHA support:

   eluafx implements an XMM server and a NATIVE client

   sluafx implements a NATIVE server but no client (and so must be used in conjunction with a remote ALOHA client, which could be **elua**, **eluax**, or **eluafx**)

   The **Aloha from Lua** document has been updated to describe these variants, which (when used together) allow both the client and the server to execute entirely in NATIVE mode from Hub RAM, using two different propellers. This

offers the fastest possible client/server execution option, achieving significantly better performance than any single propeller option.

Other Changes

1. Fix issues in the **build_all** scripts in the eLua/ALOHA demo folder (i.e. */demos/elua/aloha/build_all* and */demos/elua/aloha/build_all.bat*). Affected both Windows and Linux.

Release 8.1.1

New Functionality

1. The default Lua garbage collector has been made more aggressive, to better suit the Hub RAM constraints of the Propeller. The default is now to use the "incremental" collector with a "pause" of 110, which means it will start a new cycle when the use of memory exceeds 110% of the use after the previous collection.. Once a program is functioning correctly, in most instances the settings can be relaxed to improve performance.

Other Changes

2. The Lua "hmi" module was not being initialized correctly in some versions of Lua, and it would result in a run-time error complaining that "hmi" was a nil value. Affected the Propeller 2 only on Windows, Linux and Catalyst.

3. 2. The **p2_asm** script was not generating a listing when requested using the -l option. Affected Linux only.

4. 3. Add a note about the need to recompile **eLua** programs if the address of the secondary needs to be modified to be compiled under Linux. Affected the Propeller 2 on Linux only.

Release 8.1

New Functionality

1. A new Lua variant has been added - **eLua** supports client/server programs where the client and server execute on the same Propeller, and it can be extended by the inclusion of a new serial protocol (ALOHA) to support clients and servers executing on different propellers. Client/server programs are supported on the Propeller 2 only. See the document **Aloha from Lua** for full details.

2. The **exec** command (i.e. *exec.lua)* has been extended to allow parameter substitution anywhere in the executed script. Previously, parameter substitution was performed only when **_n** was specified as an argument to the exec command itself - this was replaced with the value of the nth parameter to the current script. This is still supported, but in addition **%n** can now be specified anywhere in the script and it will be substituted with the current value of the nth parameter to the current script. In most cases, **%n** can be used wherever **_n** was used, except where **_n** represents the name of an environment variable and not a parameter, such as if it was the name (but not the parameters) of an **if** command. For an example of this, see the **loop2**

script in the folder *demos/catalyst/core*, which is a minor variant of the **loop** script. The **exec** command is available on the Propeller 2 only.

3. A new **call** command has been added (i.e. *call.lua*) that allows a Catalyst script to be called from within another Catalyst script, which resumes the original script when the called script terminates.

   The **call** command can also be used on the command line, where it behaves effectively the same as the **exec** command - but when used WITHIN a script, **call** and **exec** behave differently - when **exec** is used, the executed script REPLACES the current script, whereas when **call** is used, the currently executing script will be resumed once the called script terminates. The **exec** command is still the mechanism to be used for looping scripts, as demonstrated in the **loop** and **loop2** scripts.

   For example, to execute the Catalyst script **script_2** from within the script **script_1**, passing it the original parameter **1** and **xxx** as parameters, and then once that completes, call it again with the original parameter **2** and **yyy** as parameters, the script **script_1** might contain lines like:

   call script_2 _1 xxx
   call script_2 _2 yyy

   If **exec** were used instead of **call**, the second invocation of **script_2** would never be executed because the first invocation would not return.

   Note that calls cannot be nested - i.e. a script can be executed from the command line that calls another script, but that script cannot then call any further scripts.

   To support the **call** capability, two additional lua scripts have been provided (*_save.lua* and *_restore.lua*) which can save and restore the current values of up to 23 parameters in a single environment variable:

_save	save parameters _0 .. _22 in a named environment variable (_SAVED_PARAMS is the default name)
_restore	restore parameters _0 .. _22 from a named environment variable (_SAVED_PARAMS is the default name)

   The **call** command is available on the Propeller 2 only.

4. The self-hosted catalina command (i.e. catalina.lua) can now be executed within a script. Previously, doing so would terminate the script.

5. A simple **echo** command has been added (i.e. echo.lua) which just echoes its arguments. For example:

   echo hello there!

   or

   echo %1 %2 %3

6. New functions have been added to the Lua **propeller** module, which are analogous to the corresponding C library functions:

   cogid, **clockfreq**, **clockmode**, **getcnt**, **muldiv64**, **locknew**, **lockclr**, **lockset**, **lockret**, **locktry**, **lockrel**

See the document **Lua on the Propeller 2 with Catalina** for details.

Note that **locktry**, **lockrel** and **muldiv64** functions are implemented on the Propeller 2 only.

7. All the HMI functions (e.g. **k_get**, **k_ready** ... **t_char**, **t_geometry** ..., **m_buttton**, **m_reset** ...) have been removed from the Lua **propeller** module and put in a separate **hmi** module. Also, **t_string** was missing and has now been added. If these functions are now undefined and cause runtime errors when called because they are nil, then they should be renamed - e.g. from **propeller.k_new()** to **hmi.k_new()** etc. All the Lua examples in the release have been updated accordingly.

8. A new example program (*ex13.lua*) has been added to demonstrate the use of Propeller locks as another possible synchronization mechanism in the Multi-Processing version of Lua (i.e. **mlua** or **mluax**). See the document **Lua on the Propeller 2 with Catalina** for details.

9. The multi-processing example 10 (*ex10.lua*) has been updated to the LED numbers appropriate for a P2 EDGE board rather than a P2 Evaluation board. Also, the example that tries to execute the maximum number of Lua threads (*ex8.lua*) has had the default reduced to 10 from 12 (which was not achievable in some cases).

10. A new example program (*args.lua*) has been added to demonstrate how a Lua script can process arguments whether executed from the command line, such as:

    args one 2.0 3

    or

    lua args.lua one 2.0 3

    Or from within another Lua script, such as:

    ```
    f = loadfile("args.lua")
    f("one", 2.0, 3);
    ```

    or

    ```
    loadfile("args.lua")("one", 2.0, 3)
    ```

    It is now possible to specify the pins and baud rates to be used for the default serial port and the blackcat/blackbox port to be specified on the Catalina command line using the -C command line option - e.g:

    ```
    -C _RX_PIN=1
    -C _TX_PIN=2
    -C _BAUDRATE=115200
    -C _BLACKCAT_RX_PIN=1
    -C _BLACKCAT_TX_PIN=2
    -C _BLACKCAT_BAUD=115200
    ```

11. There is a new library function that can be used to align the value of the system break (aka sbrk). If aligning the system break is needed, this function

should be called before any memory allocation function (i.e. malloc() etc). The function is:

```
/*
 * _align_sbrk - align sbrk to (1<<align), add offset, and report
 *               the final value (using t_printf) if requested.
 *               For example:
 *
 *               _align_sbrk(10,0,0); // align to next 1k boundary
 */
_align_sbrk(int align, int offset, report);
```

This is currently used on startup by Lua programs that can experience a lockup when loading libraries - this appears to be a Lua bug to do with the alignment of the system break. The following code resolve the bug in all currently known instances:

```
/* align sbrk to 2k boundary - Lua needs this! */
_align_sbrk(11,0,0);
```

Other Changes

1. The Serial 2 plugin was not being initialized correctly when the first function called was s2_rxcount(). In such a case the function would never return. The s2_rxcount() function worked correctly as long as it was NOT the first function called. Affected the Propeller 2 only.

2. The definition of the LOCKNEW macro in prop.h (aka propeller.h) may have led to symbol "lock" being declared undefined because the macro used it as a parameter to the _locknew() function which is neither required nor used. Affected the Propeller 1 and Propeller 2.

3. The definition of the _lockrel() function in prop2.h (aka propeller2.h) was incorrect - it defined the function as returning an int when in fact it does not return any value. Affected the Propeller 2 only.

4. The filename completion function in Catalyst would complete commands in the BIN directory without including the "/BIN/" prefix. If the intent was just to execute the command this worked ok, but if the intent was to process the file (e.g. if the command was a Lua script to be edited) then the completed filename was not correct and the file would not be found. Affected the Propeller 2 only.

5. The information in the 'Catalina Reference Manual (Propeller 2)' document regarding the configuration of the 2 port and 8 port serial plugins and how this may interact with the TTY and SIMPLE HMI options was out of date and has been updated. Affected the Propeller 2 only.

6. The size of the Openspin symbol table has been doubled (from 16384 symbols to 32768 symbols) as it was possible that some programs exceeded the original capacity. Affected the Propeller 1 only.

7. Some of the dynamic kernels were not correctly setting up the first thread block, which led to lockups when these kernels were used by programs that used pthreads, including some multi-processing Lua programs. Affected the Propeller 1 and Propeller 2.

8. The *Catalina_FAQ.txt* document was out of date. It has been removed.

Release 8.0

New Functionality

1. On the Propeller 2, Catalyst now incorporates the **linenoise** command line library, which provides enhanced command line editing functionality. This is implemented as various keyboard shortcuts:

LEFT ARROW (or CTRL B)	: move cursor left
RIGHT ARROW (or CTRL F)	: move cursor right
UP ARROW (or CTRL P)	: previous command in history
DOWN ARROW (or CTRL N)	: next command in history
HOME (or CTRL A)	: move cursor to start of line
END (or CTRL E)	: move cursor to end of line
CTRL U	: clear entire line
CTRL K	: clear from cursor to end of line
CTRL L	: clear screen
CTRL W	: clear previous word
CTRL T	: swap current and previous characters
CTRL C	: exit
CTRL D	: at start of line means exit (otherwise it means delete)
TAB	: command or file name completion

2. An enhanced Lua **REPL** (**R**ead **E**val **P**rint **L**oop) based on **lua-repl** has been added to Catalyst, which uses the 'linenoise' command line editing library. While the **linenoise** library is now built into Lua, it is not loaded by the default interactive program (i.e. **lua**). This is done to save space, since that program is primarily now used to execute Lua scripts. Instead, **lua-repl** is used to load **linenoise** and a few other plugins in a new *ilua.lua* script. So to start the new interactive Lua from the Catalyst command line, just enter the command **ilua**. This script provides the same keyboard shortcuts as those described above for Catalyst.

Other Changes

1. Payload's interactive mode has had some modifications to make its VT100 terminal emulation more closely match a real VT100. One change is in the implementation of what CR and LF do. For payload to correctly display the output of programs that send both a CR and LF, it is now recommended that **-q2** be used as the appropriate payload option (i.e. ignore the LF) whereas previously the recommended option was **-q1** (i.e. ignore the CR). Various documents have been updated.

Release 7.9

New Functionality

1. Catalina, catbind and bcc all have a new command line option. The new option (**-H**) accepts an address parameter and can be used to specify the maximum address that will be used by the heap. In all memory modes except

LARGE mode, the heap and stack share Hub RAM, with the heap growing upward from the highest used low Hub address, and the stack growing downward from the lowest used high hub address. This means they can eventually overlap, with potentially disastrous consequences.

The **-H** option allows this to be avoided, by limiting the growth of the heap. The program may run out of heap space, but this can be detected (e.g. **malloc** will return an error if there is no more space). The required amount of stack space can be determined by printing the current stack pointer at various suitable points in the program - below is a macro that uses inline PASM to do this, and a trivial program that uses it. This program will work in any memory model on any Propeller:

```
// this handy macro returns the current stack pointer
// in any memory model on the P1 or P2 …
#define SP PASM( \
"#ifdef COMPACT\n" \
"   word I16B_PASM\n" \
"#endif\n" \
"   alignl\n" \
"   mov r0, SP\n")

void main() {
printf("SP=0x%06X\n", SP);
while(1);
}
```

Suppose on a Propeller 1 it was known that the stack could grow down to 0x6000 - then it might be appropriate to specify **-H 0x6000** to prevent the heap ever growing large enough to overwrite the stack. The parameter can be specified as decimal (including an optional **k** or **m** suffix) or as hexadecimal (using the format $XXXXXX or 0xXXXXXX). For example, to ensure the heap never grows above 24k, leaving the top 8k for buffers and stack space, use a command like:

```
catalina prog.c -lc -H 24576
```

or

```
catalina prog.c -lc -H 24k
```

or

```
catalina prog.c -lc -H 0x6000
```

The **-H** option can be used on the Propeller 1 or 2. In all modes except **LARGE** mode the address refers to a Hub address. It can also be used in **LARGE** mode, where the heap is in XMM RAM, but the address refers to an XMM RAM address. This could be used (for example) to reserve an upper area of XMM RAM for other uses, such as for a buffer. However, note that the start address of the XMM RAM can vary from platform to platform, so check the **XMM_RW_BASE_ADDRESS** in the various platform configuration files.

2. New client/server support functions have been added to the libraries, and several new catapult demos have been added, to demonstrate them - see the document **Getting Started with Catapult** for details. The new demos in the *demos/catapult* folder are:

srv_c_p1.c & *svr_c_p2.c* -- demonstrate a C client with a C server.

srv_l_p1.c & *svr_l_p2.c* -- demonstrate a C client with a Lua server.

The above demo programs all use the library functions and a dispatcher which enables client/server programs to interact using the functionality that Catalina uses to interact between C programs and plugins. However, this is limited to service profiles that match the existing ones used for plugins. If these profiles are not sufficient, new profiles can be created along with a custom dispatcher - a demonstration of this is given in the folder *demos/catapult/custom* folder, using programs very similar to those above, but extended to include one new custom service profile.

Other Changes

1. An issue first noted in Catalina 7.6.1 was not fixed in all the affected kernels. In kernels without sufficient space to implement the basic floating point operations (+, -, *, / etc), an 8 byte data block is used to transfer information between the kernel and the floating point plugin. In some cases this block address was not being set up correctly, leading to the use of the 8 bytes starting at memory location 0 as the data block, which would overwrite the clock values on the Propeller 1, leading to issues with some functions that used timers. The specific kernels affected were the CMM and XMM kernels on the Propeller 1, and the XMM Kernel on the Propeller 2.

2. Catalina was not checking if there was sufficient heap space left before trying to allocate a new block of memory (e.g. via **sbrk**, **malloc**, **calloc** or **realloc**). As a result, a program could end up unable to allocate memory on the heap even though there was plenty of free space once the heap became too fragmented, because the function intended to defragment and consolidate the blocks on the free list was not being triggered as expected. Affected the Propeller 1 and 2.

   This issue affected programs that made a large number of dynamic memory allocation and deallocation calls for small randomly sized blocks of memory, because existing free blocks are re-used whenever possible, breaking them up into smaller and smaller pieces, and allocating a new block when the existing free blocks could not satisfy a request.

   The Catalina demo program most affected by this bug was Dumbo Basic, which allocates and then frees a very large number of small blocks when performing string operations, which meant the heap very quickly ended up highly fragmented. For example, the *eliza.bas* program would stop responding if a long sentence was entered.

   This issue has been fixed by (1) adding automatic detection of the maximum heap size, so that the defragment function is triggered when heap space is exhausted - this solves the problem for **LARGE** mode programs (e.g. when compiling Dumbo Basic for the P1 in **LARGE** mode), and (2) adding a new command line option (**-H**) which can be used in any mode to specify the maximum heap address to be used (e.g. when compiling Dumbo Basic in **NATIVE** mode for the P2).

Release 7.8.1

New Functionality

1. None. This release updates only the build process, to build Catalina as a 64 bit application on Windows.

Other Changes

1. Catalina is now built as a pure 64 bit application on both Windows and Linux. The only changes are those required to eliminate various gcc warnings, which is sometimes done by simply suppressing them (gcc is a bit overzealous in its warnings, and sometimes warns about C code which is correct because gcc does static checks during compilation and does not always correctly account for the actual dynamic program behaviour).

2. The document *BUILD.TXT* has been updated to include changes required to allow for building Geany on both 32 and 64 bit Windows (but building it as a 64 bit version is now the default).

Release 7.8

New Functionality

1. None. This release updates only the build process and packaging, and improves GitHub support.

Other Changes

1. Remove binaries from GitHub repository. They will remain in the Windows and Linux source/binary distributions but on GitHub they will exist as binary assets associated with each release.

2. Awka now compiles using 64 bit gcc, on both Windows and Linux.

3. Catalina now compiles using 64 bit gcc on both Windows and Linux. On Linux, Catalina is now built as a 64 bit application, but on Windows it is currently still built as a 32 bit application until further testing is completed. This will probably change in the next release.

4. Catalina no longer needs Cygwin to be installed to build under Windows. Only MINGW MSYS2 is required. However, the Cygwin DLL is still distributed with Catalina as it is still required by the Comms VT100 terminal emulator program.

Release 7.7

New Functionality

1. Catalina's main source of installation information is now the **README.md** file. The existing **README.TXT** file is retained as the history of recent changes, but refers users to **README.md** for installation and usage instructions. This is to accommodate GitHub's standard for README files.

2. The **use_catalina** script now checks for the existence of Catalina executables (specifically, the binary for catalina itself) and also whether a version of Gnu **make** is installed, and issues a warning if either one is not found.

3. A new **catalina_shortcuts** script has been added to allow Windows users to create Start Menu entries when Catalina is installed from a source other than the Windows Setup installer program (e.g. from GitHub).

4. Catalina distributions (other than the Windows Setup installer) no longer contain binaries for Gnu utility programs. The only Gnu program that is now required to build Catalina, Catalyst and various demo programs is Gnu **make** - and it is now recommended that this be installed on Windows using **winget** - e.g. using the command:

```
winget install ezwinports.make
```

Other Changes

1. Fixed some issues in the **BUILD.TXT** instructions for building Catalina from source on Linux.

2. Catalina scripts and Makefiles no longer assume that Gnu utilities (other than **make**) will be available. In particular the Makefiles now use the standard Windows utilities (**del**, **copy**, **move** etc) when executed on Windows, and the various Windows **clean_all** scripts now either use the updated Makefiles, or the Windows **del** and **rmdir** commands.

Release 7.6.3

New Functionality

1. The Catalina Catapult utility now ignores pragmas it does not recognize, but leaves them intact. Only one warning message is now issued (for the first such pragma) even if there are multiple unrecognised pragmas. This allows primary and secondary programs to use the Catalina parallelizer.

2. Four new demo programs are now included (in demos\catapult) for the Propeller 1 and 2 to demonstrate the use of the Catalina parallelizer with Catapult:

   For the Propeller 1:

   ll_p_p1.c - Demonstrates a parallelized primary program.

   ll_s_p1.c - Demonstrates a parallelized secondary program.

   For the Propeller 2:

   ll_p_p2.c - Demonstrates a parallelized primary and secondary program.

   ll_s_p2.c - Demonstrates two parallelized secondary programs.

3. The Catalina parallelizer utility now unregisters the factory kernel cogs when they are stopped via the "#pragma propeller stop" pragma. Previously these cogs were stopped but not unregistered.

4. The Catalina catapult utility now unregisters the secondary kernel cog when a secondary program terminates by exiting its function. Previously, these cogs were stopped but not unregistered.

5. On the Propeller 1, the cache cog is now registered when used. This means the registry is displayed correctly, and that this cog is correctly marked as being in use if a program wants to search the registry to locate an unused cog (normally, ANY_COG is used to start a new cog, which does not depend on whether or not the cog was registered). This was already the case on the Propeller 2.

Other Changes

1. The Catalina Geany command to build utilities did not specify that it should run in the project directory, not the current directory. The version in this release fixes this for all new installations of Geany, but it may not fix it for existing versions of Geany or existing projects.

   For existing versions of Geany, select the "Build -> Set Build Commands" menu item, and add %p to the "Working Directory" field of the "Build Utilities" command.

   For existing project files, open the project and select the "Project -> Properties" menu item, then select the "Build" tab and add %p to the "Working Directory" field of the "Build Utilities" command.

   Affected the Propeller 1 and Propeller 2 on Windows and Linux.

2. Fixed a bug that may have led to some symbols being defined multiple times in complex parallelized or multi-model programs.

   The symbols that may have ended up multiply defined were:

   > cmmd_array, lmmd_array, nmmd_array (on the Propeller 1 & 2), and CMM_LUT_LIBRARY_array, LMM_LUT_LIBRARY_array and NMM_LUT_LIBRARY_array (on the Propeller 2 only).

   Affected the Propeller 1 and Propeller 2 under Catalyst, Windows and Linux.

3. Some superseded, unused (and undocumented) functions have been removed from the threads libraries:

   > _threadstart_CMM() and _threadstart_CMM_cog()
   >
   > _threadstart_LMM() and _threadstart_LMM_cog()
   >
   > _threadstart_NMM() and _threadstart_NMM_cog()

   The following functions should be used instead:

   > _threaded_cogstart_CMM_cog()

_threaded_cogstart_LMM_cog()

_threaded_cogstart_NMM_cog()

Affected the Propeller 1 and Propeller 2 under Catalyst, Windows and Linux.

4. The example of using the Catalina Parallelizer with the Propeller version of gcc was incorrectly modified to include Catalina's "prop2.h" rather than gcc's "propeller2.h", which prevented it compiling.

Release 7.6.2

New Functionality

1. Two new catapult multi-model demo programs have been added, which demonstrate a threaded secondary program being executed from an XMM primary program (which would not support multi-threading if it was compiled as a simple XMM program). See *demos\catapult\thread_p1.c* and *demos\catapult\thread_p2.c* - note that the **build_utilities** script must be used to build an XMM loader suitable to load them.

2. The default pins used for the second serial port when the 2 port or 8 port serial plugins were in use (pins 50 & 52) conflicted with the pins used for the PSRAM on the P2-EC32MB, and may conflict with the pins used for the HyperFlash/HyperRAM add-on board, which made these plugins unusable in XMM programs. The second port has now been disabled by default by setting the tx and rx pins to **-1** on all P2 platforms.

Other Changes

1. Fixed a problem with Catapult when a secondary program required threads - e.g. the secondary pragma included **options(-lthreads)** - but the primary program did not - the secondary program was being compiled to use the threads library but was being started using the non-threaded kernel and thus would not execute correctly. Affected the Propeller 1 and Propeller 2 under Windows and Linux.

2. Fixed a bug in the XMM dynamic kernel, which broke the *start.c* program in the *demos\p2_psram* folder, which needs to load the XMM kernel dynamically. This was the only program that uses that particular version of the kernel. Also, the Makefile referred to the target directory using a relative reference, which only worked if the programs were built in the installed source tree. Affected the Propeller 2 only under Windows and Linux.

3. The 8 port serial plugin "autoinitialize" functionality, which opens any ports defined in the relevant platform file (e.g. *P2EDGE.inc*, *P2EVAL.inc* etc) was allocating buffers using static data, not local data. This meant that the plugin would not work an XMM **LARGE** program (it worked in all other memory models, including XMM SMALL programs). Affected the Propeller 2 only under Windows, Linux and Catalyst.

4. The 8 port serial plugin test program (*demos/test_serial8_2.c*) was opening ports using static data buffers instead of local buffers, and so it would not work in XMM **LARGE** mode (it worked in all other memory models, including XMM

SMALL programs). Now it uses local buffers and works in all modes. Also, the default pin numbers for the second port are now 18 (tx) and 20 (rx). Affected the Propeller 2 only under Windows, Linux and Catalyst.

Release 7.6.1

New Functionality

1. The **_clockfreq()** function now returns a default value (80Mhz on a Propeller 1 or 180Mhz on a Propeller 2) if it finds no frequency value has been set in the appropriate Hub RAM location ($0 on a Propeller 1, or $14 on a Propeller 2).

2. The catapult **STOP** macro, which is the recommended way of terminating an executing secondary function in a multi-model program, now also unregisters the cog. If the **STOP** macro is not used, the cog should be explicitly unregistered instead.

Other Changes

1. The Catapult "start" macros (i.e. **RESERVE_AND_START**, **START_RESERVED**, **START_FIXED** and **START_OVERLAY**) were always using the value **ANY_COG** for the cog to be started instead of using the value passed to the macro in the 'cog' argument. Affected both the Propeller 1 and the Propeller 2.

2. When the COMPACT kernel was dynamically loaded, the data block used to transfer data to/from the Floating point plugin was not being set up correctly, and so the first two longs of Hub RAM were being used - which would overwrite the Frequency and clock mode, and break any functions that relied on that value, such as **_clockfreq()**, **_waitcnt()** etc.

Release 7.6

New Functionality

1. Catalina Catapult is a utility intended to simplify the process of developing, debugging and maintaining Catalina multi-model programs. For details, see the document **Getting Started with Catapult**, and/or the demo programs in the *demos/catapult* folder.

2. The C library now has two overlay load functions. The existing function:

   _load_overlay() which uses the C stdio file system.

   And a new function:

   _load_overlay_unmanaged() which uses the Catalina file system.

   The new Catalina file system version takes much less Hub RAM than the previous version, and is recommended for use on the Propeller 1 for programs that do not otherwise require stdio. However, it requires the **_mount()** function to be called before any overlays are loaded. Typically, this would be called in the main program as:

```
_mount(0, 0);
```

3. The **spinc** utility now generates code that can use either the existing _load_overlay() function or the new _load_overlay_unamanaged() function depending on whether the Catalina symbol **FS_OVERLAY** is defined. For example:

   ```
   catalina -C FS_OVERLAY -lcx overlay.c
   ```

   The difference is that while the unmanaged version still needs to be compiled with the extended file system (e.g. **-lcix** or **-lcx**) it uses the Catalina file system functions, which are much smaller than the stdio file system functions. However, note that the **_mount()** function must be called before any overlays are loaded. Typically, this would be called in the main program as:

   ```
   _mount(0, 0);
   ```

4. The **spinc** utility now accepts hex values (e.g. 0x200) as arguments for the -s stack size command line option. For example:

   ```
   spinc -s 0x200 program.binary > xxx.inc
   ```

5. Catalina now accepts the definition of the Catalina symbol **P2** to mean the same as the **-p2** command line option. This allows the propeller version to be specified using **CATALINA_DEFINE**.

Other Changes

1. None.

Release 7.5

New Functionality

1. Lua has now been integrated into the Catalina library. This makes it much easier to build C programs that embed Lua. Previously to do this the whole Lua distribution had to be compiled with the C program, but now only the Lua initialization module (*linit.c*) is required because the rest of Lua can be included by simply specifying either **-llua** or **-lluax** on the Catalina command line. Examples of doing this (and an explanation of the difference between **-llua** and **-lluax**) are provided in a new demo folder called *demos/lua*. See the *README.TXT* file in that folder for more details.

   This also means that building C programs that embed Lua is now possible using the self-hosted version of Catalina on the Propeller 2. However, while this is now supported, it is not yet very practical since it takes hours to compile even a very simple C program that embeds Lua (*lhello.c*).

   Note that the Catalyst version of Lua does NOT use the compiled version in the library. This is because some of the options offered by the Catalyst version of Lua (e.g. **ENABLE_PSRAM**, **ENABLE_HYPER**) require Lua to be recompiled from source.

   Applies to Windows and Linux for the Propeller 1 and 2, and Catalyst for the Propeller 2.

2. The self-hosted version of catalina has an additional option -W which can be used to send options to the compiler (specifically, to **rcc**). For instance, to suppress warnings use the option **-W-w** which will pass the option **-w** to **rcc**. Applies to Catalyst on the Propeller 2.

3. A new include file has been added (*lut_exec.h*) which simplifies the definition of inline PASM and C code to be loaded and executed from the LUT (i.e. using LUT execution mode). Code executed from the LUT is limited to 254 longs.

   Inline PASM executed from the LUT is supported in all memory models.

   C code executed from the LUT is only supported for the **NATIVE** memory model, and has the additional limitation that it must be 'leaf' code - i.e. it cannot call any other C functions (except those that can be 'inlined' by the optimizer).

   Demos of LUT execution have been provided in a new *demos/lut_exec* folder. Both PASM and C examples are included. Applies to the Propeller 2 only.

Other Changes

1. The default Propeller 2 clock parameters are specified in each platform file (e.g. *P2_EDGE.inc*, *P2_EVAL*.inc, *P2_CUSTOM.inc* etc). These could be overridden on the command line via the **-f**, **-F** & **-E** command line options, which are used to calculate appropriate values for **_XDIV**, **_XMUL** and **_XDIVP** (see the platform include files or the Propeller specifications for more details on these values).

   The defaults in previous releases were as follows:

```
_XDIV   = 4     '\ crystal divider              to give 5.0MHz
_XMUL   = 72    '| crystal / div * mul          to give 360MHz
_XDIVP  = 2     '/ crystal / div * mul / divp   to give 180MHz
```

   These defaults have been changed in this release to:

```
_XDIV   = 1     '\ crystal divider              to give 20MHz
_XMUL   = 9     '| crystal / div * mul          to give 180MHz
_XDIVP  = 1     '/ crystal / div * mul / divp   to give 180MHz
```

   This results in the same default clock frequency (180Mhz) but the new values should mean the clock is more stable. Applies to the Propeller 2 only.

2. Eliminated errors and warnings issued when building the Catalina libraries. Most of these were because some of the libraries were being compiled for a Propeller platform (1 or 2) that did not support them. Now only the supported libraries are compiled for each Propeller platform.

3. Eliminated warnings about deprecated code when compiling **catdbgfilegen**, which is the Catalina utility that generates debugging information.

Release 7.4

New Functionality

1. Two new benchmarks (Whetstone & Dhrystone) have been added in the demos\benchmarks folder.These are used to help assess the improvements in XMM performance and also the new XMM options added in this release (see point 2 below). The existing benchmarks (*fibo.c* and *ackerman.c*) were not very useful for assessing XMM programs because they are so small they typically fit entirely in the cache, so are not representative of programs that require XMM RAM. They also did not test floating point performance.

   The benchmarks have been modified very slightly from the originals, primarily to remove the need to interact with them.

   They are applicable to both the Propeller 1 and the Propeller 2.

2. The overall XMM speed has been increased on the Propeller 2, and new XMM related compile-time options have been added that can be used to increase the speed further in specific instances. These changes are applicable to the Propeller 2 only.

   The basic speed of all XMM (**SMALL** or **LARGE**) programs has been increased by between 4% and 8% (the exact improvement depends on the program, the memory model used and the size of the cache used).

   Additional speed improvements (up to 25% in some cases) can be achieved by defining one or more of the following new Catalina symbols:

   LUT_PAGE Use the LUT to hold the current cache page and execute XMM code from there instead of from Hub RAM. The page size can be up to 1k (the second 1k of LUT is used as a common code library). The page size limit of 1k constrains the cache geometries that can be used. More details on this are given in *target\p2\constant.inc*

   LUT_CACHE Use the LUT to hold the entire XMM cache. This must be combined with **CACHED_1K**. It gives good performance with a very small cache size, and frees up valuable Hub RAM for other purposes. Only one of **LUT_PAGE** and **LUT_CACHE** can be specified.

   CACHE_PINS Use 2 pins in repository mode to communicate between the XMM kernel and the XMM cache, instead of communicating via Hub RAM. The pins used are specified in the platform configuration file (e.g. *P2EDGE.inc*) and cannot be used for any other purpose.

   FLOAT_PINS Use 4 pins in repository mode to communicate between the XMM kernel and the Floating point plugin instead of via Hub RAM. The pins used are specified in the platform configuration file (e.g. *P2EDGE.inc*) and cannot be used for any other purpose. **FLOAT_PINS** is only supported by

the Float_C plugin, which is the one loaded by default when the XMM kernel is used, or when the -lmc option is specified on the command line.

Unfortunately, no single combination of the above options gives the "best" performance in all possible circumstances - the results depend on the program, the memory model and cache size used, and how much floating point is used by the program.

In general, specifying all of **LUT_PAGE**, **CACHE_PINS** and **FLOAT_PINS** gives good results in most cases, and will typically result in a speed increase of around 10% for XMM **LARGE** programs, and 20% for XMM **SMALL** programs. A real-world example of this is that when these options are applied to the self-hosted version of the Catalina compiler itself (which is compiled in **LARGE** mode) it results in an increase in compilation speed of about 10%.

Below are some actual numbers from the benchmark programs.

First, the following common options were set in **CATALINA_DEFINE**:

```
set CATALINA_DEFINE=LARGE CACHED_64K OPTIMIZE MHZ_200 CLOCK
```

then the fibo, dhrystone and whetstone benchmarks were compiled as follows:

```
cd demos\benchmarks
catalina -p2 -lci fibo.c <plus other options - see table below>
catalina -p2 -lc -lmc whetstone.c <plus other options>
catalina -p2 -lc -lmc -D MSC_CLOCK -DCLK_TCK=1000 dhry_1.c
      dhry_2.c <plus other options>
```

This gives the following results (Release 7.3 is included in the table as a baseline):

Rel	Other Options (added using -C)	fibo (msecs)	drystone (VAX KIPS)	whetstone (KWIPS)
7.3		3064	677	877
7.4		2828	719	917
7.4	CACHE_PINS	2828	746	951
7.4	FLOAT_PINS	2828	721	980
7.4	CACHE_PINS FLOAT_PINS	2828	746	1000
7.4	LUT_PAGE	2412	671	862
7.4	LUT_PAGE CACHE_PINS	2412	710	909
7.4	LUT_PAGE FLOAT_PINS	2412	686	917
7.4	LUT_PAGE CACHE_PINS FLOAT_PINS	2412	710	970

Another result worth noting is that the **LUT_CACHE** option can be used to free up to an additional 63k of Hub RAM - i.e. it requires only 1k for the XMM cache, yet gives performance quite comparable with Release 7.3 when using 64k for the cache (as shown in the table above):

Rel	Other Options (added using -C)	fibo (msecs)	drystone (VAX KIPS)	whetstone (KWIPS)
7.4	LUT_CACHE CACHE_PINS FLOAT_PINS	3145	512	847

Note that none of these options are enabled by default because they depend on the LUT not being required for other purposes, and also on the platform having sufficient unused pins available. A good choice for the pins to use are the unused pins when the RTC add-on board is used (which uses only 2 pins out of the 8 pins the board physically occupies) - these are the defaults used in the platform files.

3. A new Propeller 1 demo program (*random.c*) has been added in *demos\random*. This program demonstrates the **RANDOM** plugin, and also demonstrates how to manually load and unload plugins at run-time.

Other Changes

4. The **_unregister_plugin()** library function was putting zero in the plugin type in the registry (which actually means **LMM_VMM** - i.e. an LMM kernel) instead of putting in the value 255 (which is **LMM_NUL**). This made no difference when a new plugin was loaded into a free cog using **ANY_COG** (which is the usual way to load plugins) but may have given the wrong results when the registry was displayed, or if the registry was searched for a free cog to load based on the plugin type in the registry. Affected both the Propeller 1 and Propeller 2.

5. The Propeller 2 PSRAM and HyperRAM demo programs in *demos/p2_ram* were broken by the addition of environment variables, which prevented some programs from building because the initialization code was too large. Affected the Propeller 2 only.

Release 7.3.1

New Functionality

4. None

Other Changes

6. When the **alloca()** function was added (release 7.0) it increased the stack requirements for all programs very slightly, but the stack sizes allocated for Factories in the Multi-Processing version of Lua was not updated. This led to various issues but the most obvious was that Lua programs that tried to use the **sbrk** function to defragment the C heap would not run correctly. Affected the Propeller 2 only.

Release 7.3

New Functionality

1. A new random number plugin has been added for the Propeller 1, based on Chip Gracey's RealRandom.spin. The plugin is enabled by defining the Catalina symbol **RANDOM**. For example:

```
catalina ex_random.c -lci -C RANDOM
```

Note that a random number generator is built into the Propeller 2, so the plugin is not required on that platform. The **RANDOM** plugin occupies a cog on the Propeller 1. It generates random data continuously, and writes a new

32 bit random number to the second long in its registry entry approximately every 100us (at 80Mhz).

C provides a pseudo random number generator called **rand()** and this must be seeded using the **srand()** function, otherwise it will always return the same sequence of pseudo random numbers.

In previous releases, the **getrand()** function was a "stopgap" function - it generated true random data on the Propeller 2, but on the Propeller 1 it only generated pseudo random data, seeded by the system clock. However, there are good reasons to have both a pseudo random number generator and a true random number generator, so with the addition of the new Propeller 1 RANDOM plugin, **getrand()** has been modified to always generate 32 bit pseudo random numbers seeded by the system clock on both the Propeller 1 and 2, and a new **getrealrand()** function has been added which returns a 32 bit true random number on both the Propeller 1 and 2 (provided the RANDOM plugin has been loaded on the Propeller 1). If the RANDOM plugin has not been loaded (which may be because there is not a spare cog available) on the Propeller 1 then **getrealrand()** uses the same technique as getrand().

The new **getrealrand()** function will generate unique random numbers in situations where the Propeller 1 previously would not, such as if a program was automatically started immediately after power up. In such situations the system clock would be reset, so the previous technique of seeding a pseudo random number generator with the system clock would always generate the same sequence of pseudo random numbers.

Here is a summary of the random number routines:

rand(), **srand()**	**rand()** generates pseudo random numbers, in the range 0 to **MAX_RAND** (32767), and **srand()** can be used to seed the random number generator
getrand()	**getrand()** generates 32 bits of pseudo random data using **rand()**. On first call it also seeds the random number generator using **srand()** and the current system clock. This avoids the need to explicitly call **srand()**
getrealrand()	**getrealrand()** returns 32 bits of random data. On the Propeller 1 this will be true random data if the RANDOM plugin is loaded, otherwise it will use the same technique as **getrand()**. On the Propeller 2 it always returns true random data.

The program *ex_random.c* in *demos/examples* has been updated to demonstrate both **rand()** and **getrand()** (which both generate pseudo random numbers) as well as the **getrealrand()** function (which generates true random numbers).

2. Dumbo Basic has been updated to allow the use of the new getrealrand() function (described in point 1, above). A new **RANDOM** option has been added to the **RANDOMIZE** statement to seed Basic's built in pseudo random number generator with getrealrand().

Here are all the **RANDOMIZE** options:

RANDOMIZE	prompt for the seed
RANDOMIZE TIMER	use the the current time as the seed
RANDOMIZE <expr>	use <expr> as the seed
RANDOMIZE RANDOM	use getrealrand() as the seed (on the P2, or when compiled with **RANDOM** on the P1) or else prompt for the seed (note this is an extension to GW Basic).

The **RND** function has been extended to allow for the generation of either pseudo random or true random numbers. Instead of just **RND**, use **RND(x)** where ...

x > 0	generates the next pseudo random number
x = 0	returns the last random number generated (pseudo or true)
x < 0	generates a true random number (note this is an extension to GW Basic, where x < 0 reseeds the random number generator - but that can also be done using the **RANDOMIZE** statement).

Dumbo Basic now enables the **RANDOM** plugin by default when built for the Propeller 1, and the basic demo programs *poker.bas*, *blackjck.bas*, *startrek.bas* and *ut-trek.bas* have been updated to use the **RANDOM** option of the **RANDOMIZE** statement instead of the **TIMER** option, which depended on variations in user input to generate a random seed.

3. Dumbo Basic's **USING** clause (used in **PRINT**, **PRINT#** and **LPRINT** statements) now accepts more GW Basic format options:

+	a '+' indicates a leading sign will be printed (i.e. '+' or '-') except as noted for '-' (below). The '+' also counts as another digit in the output. In GW Basic the '+' must be in the first position of the **USING** clause, but in Dumbo Basic it can appear in any position before any decimal point (the actual position makes no difference - the sign will always be printed immediately before the number). Note that (unlike GW Basic) Dumbo Basic always prints a sign for negative numbers - adding '+' makes it also include a leading sign for positive numbers.
-	a '-' anywhere in the clause indicates a trailing sign will be printed for negative numbers. The '-' does NOT count as another digit in the output - if the number is not negative, a space will be printed in its place. In GW Basic the '-' must be in the last position of the USING clause, but in Dumbo Basic it can be in any position (the actual position makes no difference - the '-' will always be printed immediately after the number). If both '+' and '-' are included in the same USING clause then the '+' is just treated as another digit position.
^^^^	a '^^^^' anywhere in the clause indicates an exponent of form 'ESNN' will be used, where E is the letter 'E', S is the sign of the exponent and

NN is the 2 digit exponent. The '^' characters do NOT count as additional digits - they always indicate an exponent. In GW Basic the '^^^^' must appear at the end of the **USING** clause, but in Dumbo Basic it can be in any 4 consecutive positions. The actual position makes no difference.

For example, the statements:

```
PRINT USING "##.#^^^^"; 0.001
PRINT USING "+#.#^^^^"; 0.001
PRINT USING "##.#^^^^"; -0.001
PRINT USING "+#.#^^^^"; -0.001
PRINT USING "+#.#^^^^-"; 0.001
PRINT USING "##.#^^^^-"; -0.001
PRINT USING "+#.#^^^^-"; -0.001
```

would result in the following output:

```
 1.0E-03
+1.0E-03
-1.0E-03
-1.0E-03
+1.0E-03
 1.0E-03-
 1.0E-03-
```

Other Changes

1. The pseudo random number generator **getrand()** was returning only 31 bits of random data, not 32. Affected the Propeller 1 only.

2. Dumbo Basic was not correctly keeping track of the current position in the current line of output, which meant using ',' as a separator to move to the next 14 character print zone did not always end up with the output in the correct position. Affected Dumbo Basic on the Propeller 1, Propeller 2, and also on Windows and Linux.

3. All the Dumbo Basic example programs are now UNIX format files, not DOS format files. This makes them easier to edit using vi on Catalyst, which expects UNIX format files by default. Affected Dumbo Basic on the Propeller 1, Propeller 2, and also on Windows and Linux.

Release 7.2

New Functionality

4. The sources of Dumbo Basic are now included as a demo C program for Catalyst because it can now be compiled by the Catalyst version of Catalina. To support this, some file names had to be changed to make them DOS 8.3 compatible:

   dumbo_basic.c -> dbasic.c
   tokenizer.c -> tokens.c
   tokenizer.pl -> tokens.pl (not required to compile Dumbo Basic)

Note that Dumbo basic is around 12,000 lines of C code, and therefore takes a long time to compile using the Catalyst version of Catalina. For instance, the Catalyst command:

```
catalina -v dbasic.c basic.c token.c -lcx -lm
```

takes around 2 hours and 30 minutes to complete!

5. The Catalyst version of the Propeller assembler (p2asm) has had its capacity increased to 20,000 symbols (from 5,000). This allows the self- hosted version of Catalina to be used to compile larger C programs, such as Dumbo Basic, and it also speeds up smaller compilations because the symbol table (which is a hash table) ends up less densely populated, which results in fewer hash table collisions.

   The compile times achieved by the Catalyst version of Catalina depends on many factors, including the platform, the clock frequency, the command line options specified, the libraries used, and the speed of the SD card, but some typical times (on a 200MHz P2 EDGE) in **NATIVE** mode are as follows:

C Program	Lines (approx)	Compile Time (normal build)	Compile Time (quick build)
hello	5	8 mins	5 mins
othello	500	10 mins	8 mins
startrek	2,000	40 mins	37 mins
chimaera	5,000	2 hours	1 hour 50 mins
dbasic	12,000	2 hours 30 mins	2 hours 20 mins

   Note that the quick build feature is not expected to speed up **NATIVE** mode compilations by much - quick build is primarily designed to speed up XMM **SMALL** and XMM **LARGE** mode compilations.

6. Catalina now defaults to using the name of the first input file without any extension as the output name if no output name is explicitly specified (using the -o command line option). For example:

```
catalina -p2 -c hello.c       -> generates hello.obj
catalina -p2 -lci hello.obj   -> generates hello.bin
```

   Previously, Catalina would include the extension, so the second command above would try to create *hello.obj.bin* - this worked ok when compiling for the Propeller 1, but failed when compiling for the Propeller 2, due to the differences in the assembler used (p2asm vs spinnaker).

7. The Catalyst version of Catalina now has a **-N** command line option, which disables the inclusion of **IF** commands in the generated script that test the exit code of each compilation step. This can speed up the execution of the resulting script, but it means the script will continue to execute even if an error is detected. Including the tests is useful to terminate long compilations that may take hours, but is less useful for short compilations where the user can easily terminate the script if required.

Other Changes

4. The speed of Catalysts auto-execution capability (i.e. the auto execution of commands in the files *AUTOEXEC.TXT* or *EXECONCE.TXT* on boot) has been increased. This also results in significantly faster Catalyst script execution times (which the Catalyst version of Catalina uses). Affected the Propeller 1 and Propeller 2 versions of Catalyst.

5. The Dumbo Basic **VALLEN** function was not working correctly. **VALLEN** is not a GW Basic function, it comes from Mini Basic, and was supposed to return the number of characters in the argument that a VAL statement would interpret when called on the same string argument. For example:

   ```
   VAL("1234XXXX") = 1234

   VALLEN("1234XXXX") = 4
   ```

   Affected Dumbo Basic on the Propeller 1, Propeller 2, and also on Windows and Linux.

6. The C include files were DOS format files on Windows, and Unix format files on Linux. This was ok on both of those platforms, but it caused problems with Catalyst - when Catalyst was built on Windows it ended up with Windows format include files when it was expecting Linux ones. To fix this, all include files are now in Linux format files on all platforms. Affected the Propeller 2 version of Catalyst only.

7. The Catalyst version of the Propeller 2 Assembler (**p2asm**) now dynamically allocates its Symbol Table array. This makes no functional difference, but it significantly reduces the size of the p2asm binary. Affected the Propeller 2 version of Catalyst only.

8. The Catalyst version of Catalina now only includes only the generic and Propeller specific code generation back-ends, and not Alpha, MIPS, OSX or X86 etc. This significantly reduces the size of the rcc binary. Affected the Propeller 2 version of Catalyst only.

9. The Catalyst version of Catalina now deletes only the output files before the compile if untidy mode is specified, not any of the intermediate files. Affected the Propeller 2 version of Catalyst only.

10. The Catalyst version of Catalina now correctly produces a *.obj* file if the **-c** command line option is specified, instead of a *.s* file. Affected the Propeller 2 version of Catalyst only.

11. The Catalyst version of Catalina was incorrectly invoking the Spin preprocessor (**spp**) on *.obj* files. Affected the Propeller 2 version of Catalyst only.

12. Dumbo Basic's PRINT statement was printing floats with precision zero unless a USING clause was specified - this bug was introduced in release 7.1.1. Affected both the Propeller 1 and Propeller 2 on Catalyst, Windows or Linux.

13. The code Catalyst uses to detect if an executing script is to be terminated (which prompts with "Continue auto execution (y/n)?" if it detects a key held down when it executes each line of the script) worked ok if the answer was **n** but did not continue correctly if the answer was **y**. Affected Catalyst on the Propeller 1 and Propeller 2.

Release 7.1.2

New Functionality

1. Catalina's VT100 terminal emulator (*comms.exe*) now accepts setting the Terminal Emulator **LockScreenAndView** option from the command line, using the option:

   /[No]LockScreen

   Also, this option is now disabled by default. Previously, it was enabled by default and had to be disabled via the Options->Advanced menu.

2. Dumbo Basic's assembly language interface has been updated to support the Propeller 2 in XMM **SMALL** and **LARGE** mode, and add better support for **COMPACT** mode. Also, a better method of compiling PASM programs to be included in Basic programs has been added (see *example.pasm* for details). Also, specific examples to demonstrate using **NATIVE**, **COMPACT** and **LARGE** modes on the Propeller 2 have been added (see *ex_call1.bas*, *ex_call2.bas* and *ex_call3.bas*) and **LARGE** mode on the Propeller 1 (see *ex_call4.bas*).

3. Dumbo Basic's **USING** clause (used in **PRINT**, **PRINT #** and **LPRINT** statements) now matches GW Basic much more closely. The main difference remaining is that Dumbo Basic always prints a '-' sign for negative numbers, whereas GW Basic does not print the '-' sign unless a '+' appears in the USING clause, and does not print a leading sign when a '$' sign is specified. Catalina's approach is much more useful and less error prone, but the option to exactly duplicate GW Basic functionality may be added in a later release.

4. Two new basic demo programs have been added - *poker.bas* and *blackjck.bas*. These have been modified from the original sources to correct various errors in the original programs, and should run on any Propeller 1 or 2 platform that supports Dumbo Basic.

5. Various improvements in the Dumbo Basic example programs. The originals were often straight out of the GW Basic manual, but were not particularly user friendly. Also, the example programs (and data files) intended only to demonstrate specific Basic statements are no longer included in the Catalyst image. Only complete Basic demo programs (such as eliza.bas and startrek.bas) are now included. The others can be manually copied to the SD card if required. Note also that executing Basic programs on the Propeller 1 requires XMM RAM, and the C3 may not have enough to run the larger demos (however, the C3 CAN run eliza.bas or the new poker.bas and blackjck.bas - albeit slowly, due to the unique implementation of XMM RAM on the C3!).

Other Changes

1. The Catalina Reference Manual for the Propeller 2 has been updated to include a description of the NMM Kernel. Affected the Propeller 2 only.

2. Dumbo Basic's **PUT** statement was not correctly extending the file when a record was PUT that was beyond the current end of the file. Affected both the Propeller 1 and Propeller 2 on Catalyst. This was due to a limitation of DOSFS and while it has been fixed in Dumbo Basic, it may affect other programs, so a note has been added to the Catalin Reference manuals in the section on File System support.

3. Dumbo Basic's OPEN statement was not using the correct file mode for **R** or **RANDOM** files. One consequence of this was that the file would not be created if it did not already exist. Affected both the Propeller 1 and Propeller 2 on Catalyst, Windows or Linux.

4. Dumbo Basic's **CALL** statement was not working correctly - an attempt to use it would result in a "Bad subscript" error. Affected both the Propeller 1 and Propeller 2 on Catalyst.

5. Dumbo Basic was not allowing a line number to follow a **THEN** or **ELSE** clause, which is permitted in GW Basic. It would only allow a line number after a **GOTO** clause. Affected both the Propeller 1 and Propeller 2 on Catalyst, Windows or Linux.

6. Dumbo Basic was failing to correctly parse some complex **IF** statements that included multiple statements on the same line separated by colons, or when IF statements were nested. Affected both the Propeller 1 and Propeller 2 on Catalyst, Windows or Linux.

7. Dumbo Basic now correctly detects if there are extra tokens on a Basic line when it is expecting an end of line. Previously, it would either ignore such tokens, or possibly get into an infinite loop attempting to process them. Now it correctly reports "Unexpected character follows statement". Affected both the Propeller 1 and Propeller 2 on Catalyst, Windows or Linux.

8. Dumbo Basic's token parser was not recognizing the **VAL** keyword. Affected both the Propeller 1 and Propeller 2 on Catalyst, Windows or Linux.

Release 7.1.1

New Functionality

1. The *startrek.bas program* has been updated to better use a VT100 terminal, and also to randomize the game on startup, based on how long the user takes to respond to an initial prompt to start the game.

Other Changes

1. Dumbo Basic had a bug in the square root (**SQR**) function - it would throw an error if the argument to the **SQR** function was zero (which is a valid argument). Affected both the Propeller 1 and Propeller 2 on Windows or Linux.

2. The startrek.bas BASIC program (*startrek.bas*), and also the C version of the same program (*startrek.c*) contain a bug which appears to have been inherited from the original basic source code and which is not compatible with GW Basic (and therefore not compatible with Dumbo Basic). Affected both the Propeller 1 and Propeller 2 on Windows or Linux.

Release 7.1

New Functionality

2. The Catalyst (i.e. self-hosted) version of Catalina now supports compiling XMM **SMALL** and XMM **LARGE** programs on the Propeller 2. Just add **-C SMALL** or **-C LARGE** to the catalina command, or add the symbol **SMALL** or **LARGE** to the **CATALINA_DEFINE** environment variable. Note that XMM compiles take longer than **NATIVE**, **COMPACT** or **TINY** compiles, so it is recommended that Quick Build (see point 2, below) be used when possible. For example, to compile the othello.c demo program on the Propeller 2 in **LARGE** mode, you could use a Catalyst command such as:

```
catalina -v othello.c -lci -C LARGE
```

3. Add support for Quick Build, which means in cases where the target is built separately to the program (i.e. the XMM **SMALL** and **LARGE** memory models) then if a target file of the correct name already exists, it is used instead of building it. Quick Build is only supported on the Propeller 2.

   For **LMM**, **CMM** or **NMM** programs, enabling Quick Build changes the compilation process to build a separate target and program file instead of doing the usual monolithic compilation. This can speed up the compilation process, but it makes such binaries 64 kb larger than normal. It is therefore mainly recommended on the self-hosted version of Catalina.

   Care needs to be taken that any existing target file is appropriate for the current compilation. If in doubt, simply do not enable Quick Build and Catalina will not use any existing target files.

   Quick Build is enabled by adding the command line option **-q** or defining the Catalina symbol **QUICKBUILD**. Enabling Quick Build also means the target file is left after the current compilation completes, rather than being deleted. Note that a target file may also exist because the -u (untidy) option was used in the previous compile.

   Things that affect the target file is anything other than the compiled C program code itself (plus any C library code that code uses, except as noted below). This includes:

   - the platform selected (e.g. via **-C C3**, **-C P2_EDGE** etc)

   - the plugins selected (e.g. via **-C CLOCK**, **-C RTC**, **-C VGA**, **-C TTY** etc)

   - the clock selected (e.g. via **-f**, **-e** and **-E**, or **-C MHZ_200** etc)

   - the memory model selected (via **-x** or **-C TINY**, **-C NATIVE**, **-C COMPACT**, **-C SMALL** or **-C LARGE**)

- the floating point option selected (via **-lm**, **-lma**, **-lmb** or **-lmc**)

- whether **NO_PLUGINS**, **NO_ARGS** or **NO_ENV** is specified

- whether multi-threading is selected (via **-lthreads**)

- whether an SD Card plugin is selected (via **-lcx**, **-lcix** or **-C SD**)

- whether debugging is selected (via **-g** or **-g3**)

- using custom memory sizes and/or addresses (via **-M**, **-P** or **-R**)

If any of these options need to change, do not use Quick Build. The results will be unpredictable.

4. The interpretation of the **-v** and **-d** options in catalina, catbind and bcc has been made more consistent. These options are now completely disjoint, and now neither one implies the other, whereas previously **-d** implied **-v**. The types of messages enabled by these options is now as follows:

   -v enables messages that indicate WHAT the compiler is doing.

   -d enables messages that indicate HOW the compiler is doing it, or that show intermediate results which may assist in diagnosing what happened when something goes wrong.

   To see all the messages that were previously output by **-d**, use BOTH **-d** and **-v**.

   Using **-v -v** does not now display any additional messages, but there are still cases (e.g. in bcc and binbuild) where using multiple **-d** options enables more detailed diagnostic output.

Other Changes

1. The Spin Pre-Processor (spp) was also processing C style comments,so it was treating the Spin *//* operator as if it was introducing a comment and truncating the rest of the line. Did not affect the use of spp by Catalina (which never generated this operator), but may have affected Spin/PASM programs if used stand-alone, Affected Windows, Linux and Catalyst.

2. Minor documentation updates concerning specifying the memory layout using the **-x** option, and additional documentation of the Propeller 1 **-x10** option, which means use the **COMPACT** memory model and kernel, and the **SDCARD** loader. Affected both the Propeller 1 and Propeller 2 on Windows or Linux.

3. The file *README_P2.TXT* in the Catalina directory contains some spurious characters where double quotes should have appeared. Affected Windows and Linux.

4. In the self-hosted version of Catalina, the **-B** command line switch (for baud rate) was not resulting in the correct C symbol definition. Affected The Propeller 2 on Catalyst only.

5. The multicog 'dynamic_kernel.c' has been updated to use pin 38 instead of pin 0 when built for the P2_EDGE platform. Affected the Propeller 2 only.

6. Tidied up the structure of the Propeller 2 target files, to eliminate unnecessary duplication. The plugin support code is now in one file (*plugsup.inc*) and the plugins themselves are included in another file (*plugins.inc*). Affected the Propeller 2 only.

7. The Catalina binder (**catbind**) now uses external utilities for the binary build and statistics commands (see *binbuild.c* and *binstat.c*).

Release 7.0

New Functionality

5. Catalina now supports the **alloca()** function. This function is supported in all memory models, and on both the Propeller 1 and Propeller 2. Note that alloca is not part of the ANSI C standard, but it is commonly available in C compilers.

   For an overview of alloca, see ***https://linux.die.net/man/3/alloca***

   The main advantage of alloca is that it is a very efficient alternative to malloc. It is particularly suited to C on the Propeller, where the heap and stack can use different types of RAM. The C language does not cater very well for architectures that can have different types of RAM.

   Note that there are limitations of using alloca, which are described in the link above - the main one is that (unlike memory allocated via malloc) memory allocated via alloca must not be referenced once the function in which it was allocated returns to its caller. However, it is worth remembering that such memory allocated in the C main function will remain valid for the duration of the program. But this also illustrates another limitation of alloca - which is that there is no way to DE-allocate such memory once allocated other than by exiting the function in which it is allocated.

   To see a situation in which alloca is particularly useful on the Propeller, consider an XMM LARGE program. In these programs the stack is in Hub RAM, but the heap is in XMM RAM, and the malloc functions always operate only on the heap. Prior to having alloca available, there was no simple way of allocating an arbitrary amount of Hub RAM. While Catalina release 6.5.5 added hub_malloc functions specifically to do this, these functions (like malloc) are large in code size and also very slow when compared to alloca.

   For an example of using alloca, see the demos\alloca folder.

   NOTE: the reason for releasing this version of Catalina as 7.0, rather than 6.x is that implementing alloca efficiently required a minor kernel change, which means that code compiled with Catalina 7.x MUST NOT BE COMBINED with code compiled by Catalina 6.x or earlier. For instance, object files or libraries compiled with Catalina 6.x must be recompiled before using them with Catalina 7.x.

Other Changes

8. The Posix threads functions were returning the correct error values, but not setting the system errno variable, so programs that used errno rather than the function return value, or used the perror function to print the error, may not have detected or printed the error correctly. Affects both Windows and Linux. Affects the Propeller 1 and Propeller 2.

9. The hub_malloc functions introduced in Catalina 6.5.5 did not work on the Propeller 1. In this release they also now work on the Propeller 1 for programs compiled in XMM LARGE mode (which is the only mode in which they offer any benefit over the normal malloc functions). Affects both Windows and Linux. Affect the Propeller 1 only.

10. Minor updates to the Catalyst build scripts to prevent building Catalyst programs that are not supported on the Propeller 1, such as the real-time clock programs. Affects both Windows and Linux. Affects the Propeller 1 only.

11. Various changes in this release have had minor impacts on the code size, execution speed and stack requirements for C code. For example, the default stack size allocated to a posix thread has been increased from 100 longs to 120 longs, and the default stack size used by the spinc utility has been increased from 80 bytes (20 longs) to 100 bytes (25 longs). If the stack size is manually specified it may need to be increased slightly in a similar manner. This is true for stack space allocated to any C code executed using Catalina's multi-processing support (i.e. multi-threading, multi-cog or multi-model support). If a program that uses multi-processing worked under a previous release of Catalina but does not work correctly in this release, try a small increase in the size of any stacks manually allocated. Affects both Windows and Linux. Affects the Propeller 1 and Propeller 2.

Release 6.5.5

New Functionality

6. An additional implementation of malloc has been added to the library which always allocates from Hub RAM. It is intended to be used primarily in XMM LARGE programs, where the normal malloc allocates from XMM RAM. In other memory models, including XMM SMALL programs, the normal malloc allocates from Hub RAM anyway, so this new hub version would not provide any benefit.

   The following functions are defined in a new include file (*hmalloc.h*), which mirror the usual malloc related function definitions (defined in *stdlib.h*):

```
void   *hub_calloc(size_t _nmemb, size_t _size);
void   hub_free(void *_ptr);
void   *hub_malloc(size_t _size);
void   *hub_realloc(void *_ptr, size_t _size);
```

   To support the hub malloc functions, there are also new functions **`hbrk()`** and **`_hbrk()`** which are analogous to the usual **`sbrk()`** and **`_sbrk()`** functions, but which allocate from Hub RAM instead of XMM RAM.

An example of using the new malloc is provided in "demos\hub_malloc". The program provided can be compiled to use either standard malloc or the new hub malloc. See the README.TXT file in that folder for more details.

An XMM LARGE program can use both malloc and hub_malloc freely, but other programs should use only one or the other. Also, note that some library functions (notably the stdio and posix thread functions) use malloc internally, and so programs other than XMM LARGE programs should not use the hub malloc functions unless the library is first recompiled to ALSO use hub malloc - this can be done by defining the Catalina symbol HUB_MALLOC when compiling the library. This may speed up XMM LARGE programs that make very heavy use of stdio.

Note that using the hub malloc functions reduces the amount of stack space (which is always in Hub RAM in every memory mode) available to the program.

For an example of using HUB_MALLOC to compile the library, see the **build_all** scripts provided in the folder *demos\catalyst\catalina*. Using this option has been tested, but since it reduces the available Hub RAM for ALL programs it is not enabled by default.

Note that you can use the two types of allocated memory freely without knowing what type of memory it is, but that memory allocated using the normal malloc functions MUST be re-allocated or freed using realloc and free, and memory allocated using the hub malloc functions MUST be reallocated or freed using hub_realloc and hub_free.

7. The Catalyst time utility has been updated to allow the time to be set using local time (the previous versions required it to be set using UTC time, which can still be done provided the -u or -w flag is specified).

   Note that to set time using local time, both time AND date must always be specified, since otherwise the program does not know whether or not to adjust the local time for daylight savings.

   The following are acceptable ways to set the date and/or time:

```
time -u 11:11:12 PM           -- set UTC time only (12 hour)
time -w 23:11:12              -- set UTC time only (24 hour)
time -u 01/01/2000            -- set UTC date only
time 01/01/2000 11:11:12 PM   -- set both LOCAL date and time
time 01/01/2000 23:11:12      -- set both LOCAL date and time
```

8. A new lua script is included with Catalyst called *attrib.lua*. This script allows the current text attributes (i.e. effects and color) to be specified on the Catalyst command line when using a serial HMI option and a terminal emulator.

   The syntax of the command is:

```
attrib [-d] [ [effect ...] [ color ] [ on ] [ color ]
```

   Here are some examples:

```
attrib yellow                          -- yellow fg
attrib on blue                         -- blue bg
attrib underscore                      -- set underscore effect
Only
attrib blink blue                      -- set blink and blue fg
attrib red on green                    -- red fg on green bg
attrib red green                       -- same as previous
attrib reverse bright red on green -- green fg on bright red bg
attrib normal                          --  reset  effects  and  fg  and  bg
colors
```

However, note that which text attributes are supported and what they will do depends on the terminal emulator in use, and not on this program. For instance, Catalina's external VTxx emulators do not support italic, and they interpret bright as meaning both bright and bold. Catalina's payload internal terminal emulator does not support setting any attributes at all.

9. The Lua scripts find.lua, freq.lua and wild.lua are now in the folder demos/catalyst/core, with all the other Lua scripts used by Catalyst. The find.lua and freq.lua now both support a -i flag to specify that they should be case insensitive. For example:

```
find -i print *.bas    -- find 'print', 'PRINT', 'Print' etc

freq -i *.txt          -- count word frequency ignoring case
```

10. The Comms program "paste" functionality has been improved (note that this change does not affect Catalina itself, only the Comms terminal emulator).

    Previously, it was easy to paste text that would overflow either the program's own paste buffer (which was only 256 characters) or else it would overflow the keyboard buffer in the recipient program. Now, the size of the paste buffer is configurable, and after sending every group of characters (default is 15, but this is also configurable), the program waits until the recipient stops sending characters back - however, the program assumes that more than just the pasted characters will be sent by the recipient - e.g. in the case of text editors such as "vi" that use DEC escape sequences, the number of echoed characters may be much, much larger than the number of character sent, so the program cannot just check that the character itself has been echoed, or wait for a certain number of characters - it must wait for some time after each character to see if it has received the last one (if any) before sending more. This time is also configurable.

    To configure the paste functionality, three new command-line options have been added to the Comms program:

    ```
    /PasteSize=nnn

    /PasteGroup=nnn

    /PasteDelay=nnn
    ```

    PasteSize is the maximum size of the paste buffer. Default is 2048. This represents the largest number of characters that can be pasted in each single paste operation.

PasteGroup is the maximum number of characters sent in each group when processing the paste operation. The default is 15, to accommodate serial HMI options with small keyboard buffers (such as TTY). If the recipient program has an even smaller keyboard buffer, this number can be reduced to as low as 1, in which case the program will wait after each character sent.

PasteDelay is the delay in milliseconds to wait to ensure the last character has been received after sending each group. Default is 250. At low baud rates, this number can be increased, but if the **PasteGroup** size is reduced, it may be able to be reduced.

The default values have been selected to perform well when using the Comms program to talk to the Catalyst **vi** text editor on a Propeller 2 at 230400 baud, using either the TTY or SIMPLE serial HMI options. They may need to be tweaked in other circumstances (e.g. when using a Propeller 1 at 115200 baud, it may be necessary to both reduce **PasteGroup** to 8 and increase **PasteDelay** to 500).

Other Changes

3. The Propeller 2 version of the sd card plugin (cogsd.t) did not work with Sandisk SD cards which do not implement CMD1 correctly, such as some newer Sandisk Extreme cards. This has been fixed. The version of the SD plugin in this release should work with all SD cards.

   However, note that for Sandisk Ultra cards it is recommended not to boot the Propeller 2 from the SD card itself - instead, boot the Propeller 2 from FLASH (e.g. load Catalyst into FLASH and boot it from there). Then the SD card access works reliably even on these SD cards.

   Affect both Windows and Linux. Affects the Propeller 2 only.

4. The **_set_int_X()** functions (where X = 1, 2 or 3) did not take into account that EACH COG has independent interrupts, so if the same functions were called from multiple cogs they were overwriting the interrupt configuration data, so only the last such interrupt would work. This has been fixed by allocating enough space to store the interrupt configuration for each cog separately.

   Affect both Windows and Linux. Affects the Propeller 2 only.

Release 6.5.4

New Functionality

11. Catalina now issues an error message if interrupt code is included in a program compiled in XMM (SMALL or LARGE) mode.

12. A new multi-model demo has been added for the Propeller 2 which shows how a primary program (which can be an LMM, CMM, NMM or XMM program) can execute a secondary program (which can be an LMM, CMM or NMM program) that uses interrupts. Also, the Makefile has been updated to build the XMM demos on Propeller 2 platforms that have XMM RAM.

Other Changes

5. The C interrupt library functions **`_clr_int_X()`** - (where X = 1,2,3) - did not compile correctly for programs compiled in COMPACT or TINY mode. Affected the Propeller 2 only. Affected both Linux and Windows.

6. The multi-model support for the Propeller 2 had an issue if the primary program was an XMM LARGE program (XMM SMALL programs worked correctly). Affected the Propeller 2 only. Affected both Linux and Windows.

Release 6.5.3

New Functionality

13. The default behaviour of the Catalina Geany "Catalina Command Line" menu entry (on the Build menu) is to open the window in the current project directory instead of the user's home directory. Affects Windows only (on Linux, Geany already did this).

14. When using fymodem, if no filename is specified to receive, the filename in the initial received packet is used but not the path, which means the received file will always end up in the current directory. This resolves the Windows issue that arose when specifying a file name to send using a path as well as a file name, since Windows paths and Catalyst paths use different path separator characters (i.e. '\' vs '/' respectively).

Other Changes

7. The C interrupt library functions **`_clr_int_X()`** - (where X = 1,2,3) - did not compile correctly because they used **`iretX`** instead of **`retiX`**. Affected the Propeller 2 only. Affected both Linux and Windows.

8. Fixed a bug in the dynamically loaded XMM Kernel which meant that SMALL mode programs may not have executed correctly if the kernel was loaded dynamically. Affected the Propeller 2 only. Affects both Windows and Linux.

9. A spurious **`drvl #57`** debug instruction was left in the RTC clock plugin. Affected the Propeller 2 only. Affected both Windows and Linux.

10. On the Propeller 2, the XMM Kernel now has the floating point comparison function internal to the kernel to improve performance. Affects the Propeller 2 only. Affects both Windows and Linux.

11. On the Propeller 2, the XMM Kernel now uses **XMM_ReadLong** and **XMM_WriteLong** to read and write longs instead of **XMM_ReadMult** and **XMM_WriteMult** to improve performance. Affects the Propeller 2 only. Affects both Windows and Linux.

12. The Propeller 2 cache API now includes two new page functions for reading and writing pages as longs rather than bytes, provided the buffers are long aligned and the length of the buffers is a multiple of 4. These new functions are called **XMM_ReadLongPage** and **XMM_WriteLongPage**. This will potentially improve performance on future Propeller 2 platforms, but is already the case on current XMM API implementations (i.e. PSRAM on the P2 EVAL

and P2 EDGE boards). Affects the Propeller 2 only. Affects both Windows and Linux.

Release 6.5.2

New Functionality

1. Catalina now generates debugger information (i.e. .stabs) for structures that contain pointers to themselves or to types that are forward defined, and Catdbgfilegen now knows how to process such .stabs.

2. Blackbox now has the following usability enhancements:

   - The printing of the hex values for 1 and 2 byte variables is now more consistent - previously a value like -1 would have been printed as **-1 (0xFFFFFFFF)** even if the type was a 1 or 2 byte variable. Now, these will be printed as **-1 (0xFF)** or **-1 (0xFFFF)** respectively.

   - Values specified with a size that is not 4 bytes, such as **byte**, **char**, **word** or **short** now mask the value to the correct size, so **byte -1** is **0xFF** and not **0xFFFFFFFF**, and **short -1** is now **0xFFFF** and not **0xFFFFFFFF**.

   - The printing of character values now always includes the decimal value and the character value (if it is printable). For example, if variable c is a char that contains the value 'x' then it will be printed as **120 ('x' 0x78)**. If it contains the non-printable character ESC then it will be printed as **27 (0x1B)**.

   - If the verbose mode is **on**, then variables will be printed with type and location information as well as their current value. For example, if verbose mode is **off** the command **print i** might print:

     ```
     = 1 (0x00000001)
     ```

     but if verbose mode is **on** it might print:

     ```
     LOCAL i int <size=4,align=4> @ register 23
     int @ register 23 = 1 (0x00000001)
     ```

   - Values used in commands such as read, write and update can now include simple scalar variables or the address of a variable, so it is now possible to say things like:

     ```
     update i = j
     update k = &i
     read i
     read xmm &i
     write hub &i = j
     ```

     However, note that the read and write commands ALWAYS read and write 4 bytes, even if the expression is a 1 or 2 byte value, and even if the address being written is the address of a 1 or 2 byte variable.

- It is now possible to use size specifiers to convert values, so the following are all valid no matter what the type of variable x:

```
update x = char 'a'
update x = short 65535
update x = float 1.234
```

- The type of a variable was being shown as **<anon>** in some cases, but now either the correct type is printed or no type is printed.

Other Changes

1. Programs that use the new Real-Time Clock (RTC) plugin would not compile if the **-g** -or **-g3** options were also used (to use the Blackbox debugger) and the program was compiled in **NATIVE** or **COMPACT** mode. Affected both the Propeller 1 and Propeller 2 on Windows or Linux.

2. Payload previously detected if an extension was present by looking for "." in the file name but this causes problems in Geany, which uses the full path name (which may contain a "."). Also, payload now tries the base file name BEFORE trying any extensions, which means it works correctly in Geany if there are both .bin and .binary files in the project folder.

Release 6.5.1

New Functionality

1. Added support for reading and writing XMM RAM on the Propeller 2, and for specifying Propeller 2 specific memory models (e.g. on the command line or using the xmm command).

2. Added support for (limited) indexing into arrays. The index must be the last element specified, so

   `array[index]` is supported

   `array[index].field` is **NOT** supported

3. Added support for (limited) use of the '->' operator on pointers. The operator is only supported when the first element specified is a pointer. This operator and can only be followed by a single field (including optional index), so

   `p->elt` is supported

   `p->elt[index]` is supported

   `p->elt1.elt2` is **NOT** supported.

4. Added support for (limited) use of the '*' operator when the first element specified is a pointer, so if p is a pointer then

   `*p` is supported

   Also, note that '*' can be combined with '->' so if p and q are BOTH pointers then

`*p->q` is supported

and means `*(p->q)`

5. Added support for variable names, &variable and &function being used to specify the address in read and write commands.

Other Changes

1. The Blackbox debug cog was always resetting pins 62 & 63 even if it was configured to use a different serial port. Affected Windows and Linux, Propeller 2 only.

2. Fixed an issue that meant word-sized values (e.g. short or unsigned short) in structures were not displayed correctly. Affected Windows and Linux, Propeller 1 and Propeller 2.

3. Fixed an issue with determining the offset of a field within a nested structure or union. Affected Windows and Linux, Propeller 1 and Propeller 2.

4. Fixed an issue that meant negative values could not be written using the write command. Affected Windows and Linux, Propeller 1 and Propeller 2.

5. Fixed an issue with the type search, which was aborting the search at the first instance of "<array>" (indicating an anonymous array) instead of reporting "not found" when searching for such types. This led to Blackbox being unable to determine the type of variables in some instances. Affected Windows and Linux, Propeller 1 and Propeller 2.

Release 6.5

New Functionality

6. Catalyst now uses the value of the **PROMPT** environment variable as the prompt if it exists (Propeller 2 only). If not, the default prompt of "> " is used. The prompt can be up to 20 characters, and can be set from the Catalyst command line using the "set" command. For example:

   ```
   set prompt="Catalyst> "
   ```

   Environment variables are supported on the Propeller 2 only.

7. Catalyst now uses the **LUA** and **LUAX** environment variables to determine the executable to use to execute normal and compiled Lua scripts on the Propeller 2. These environment variables can be set on the command line using the "set" command. If not set, then "lua" and "luax" are used by default to execute files with a .lua or .lux extension (respectively).

   For example, to execute ".lua" files using the XMM LARGE version of Lua, and ".lux" files using the Multi-threading version of Lua:

   ```
   set LUA=xl_lua
   set LUAX=mluax
   ```

   Environment variables are supported on the Propeller 2 only.

8. The value of the **LUA_INIT** environment variable is now used by Lua on the Propeller 2 to execute Lua commands before entering interactive mode. This environment variable can contain either a Lua statement to execute, or the value "@filename" to indicate that the statements should be read from a file with name "filename". This environment variable can be set on the command line using the "set" command. For example:

   ```
   set LUA_INIT="_PROMPT='Lua> '"
   ```

   or

   ```
   set LUA_INIT="@LUA_INIT"
   ```

   Environment variables are supported on the Propeller 2 only.

9. The lua scripts list.lua, find.lua, freq.lua and wild.lua have been updated to be more consistent with other Catalyst commands. Primarily minor cosmetic improvements but note that the order of parameters of the find script has been reversed, and that a command can now be specified for the wild script to use. The various syntaxes are now:

   ```
   list [ file_spec ]
   freq [ pattern ] file_spec
   find pattern [ file_spec ]
   wild [ command ] file_spec
   ```

10. Payload now accepts '_' as well as ',' to separate the row and columns specified in the **-g** command-line option. For example, both the following will specify the same window geometry when an interactive terminal window is requested (i.e.80 columns and 40 rows):

    ```
    -g 80,40
    -g 80_40
    ```

11. This change allows the -g option to be specified in the **Loader** field of the Project Properties in a Geany project file - previously it could not because the comma caused Geany to misinterpret the options. To facilitate this, the default Geany **Download and Interact** build command is now **payloadi** instead of **payload**. This means that the terminal window this command opens will be sized correctly when geometry is specified using an option of the form **-gXXX_YYY**.

Other Changes

1. Catalina's version of the Geany IDE crashed if there was no current project and a file was opened (e.g. using the **File->Open ...** menu entry or pressing the **Open** button). Affected Windows and Linux.

2. The Linux version of Catalina had some script files that were in DOS format instead of Unix format, which meant they could not be executed by the bash shell. Affected Linux only.

3. Some required DLLs were missing, which meant some executables (such as payload) may have reported missing DLLs if they could not be found on the user's path (e.g. if MSYS2 was not installed). Affected Windows only.

4. The script that starts the Windows Geany IDE now uses the Windows START command, which means the Geany window always starts visible - previously it would sometimes start the Geany window minimized. Affected Windows only.

5. The Basic demo *ut-trek.bas* did not execute correctly because Dumbo Basic was not correctly initializing array variables to zero. Also, there are some minor cosmetic changes in the program to improve the format of the program output. Affected the Propeller 1 and 2 on Windows and Linux.

Release 6.4

New Functionality

1. Catalina and Catalyst now both support environment variables on the Propeller 2. Previously, the C **`getenv()`** function would only return hardcoded values for specific environment variables that needed to be present to meet the ANSI C standard, and this is still the case on the Propeller 1. But proper support has been added for setting and fetching environment variables from C programs, Lua programs and on the Catalyst command line when using the Propeller 2 and the Catalyst program loader.

   There are two new functions provided (defined in *stdlib.h*) that affect environment variables:

   ```
   int setenv(char *name, char *value, int overwrite);
   int unsetenv(char *name);
   ```

   These work as described in Linux (e.g. see **man setenv**), but with the limitations described in the notes below.

   Two corresponding functions have been added to the Lua **propeller** module:

   ```
   propeller.setenv(name, value, overwrite)
   propeller.unsetenv(name)
   ```

   The corresponding **`getenv(name)`** function already existed in Lua, but it is in the **os** module:

   ```
   os.getenv(name)
   ```

   See also the description of the new Catalyst **set** command, below.

   Notes:

   - Environment variables are stored in the file *CATALYST.ENV* in the root directory of the SD Card. This is a text file that can be edited with any text editor (such as **vi**), but unless the format of the file is correct, the results of **`getenv()`** and **`setenv()`** are not defined. Only the first 2048 characters of this file are used. If the definition of an environment variable goes beyond that limit, it will be truncated.

- On the Propeller 2, the CogStore plugin (used by the Catalyst program loader to implement command-line arguments) now also supports storing up to 2048 characters in the LUT. This is used to pass the environment to programs started by Catalyst, and this is why environment variables are limited to a total of 2048 characters.
- To be able to retrieve the value of an environment variable, a C program must be started using the Catalyst program loader. However, the program does not need to be compiled with the extended C library, since the environment is passed using CogStore. However, to *set* an environment variable, a C program must be compiled with the extended C library (i.e. **-lcx** or **-lcix**).
- Setting an environment variable does not affect the *current* environment. The variable is set by writing to the *CATALYST.ENV* file, and will be available to C or Lua programs *subsequently* loaded by Catalyst.

Environment variables are currently supported on the Propeller 2 only.

2. A new Catalyst **set** command has been added. This command is supported on the Propeller 2 only. It allows environment variables to be displayed or set from the Catalyst command line, or in Catalyst scripts. The syntax of the command is:

   set [options] [NAME [= [VALUE]] ...]

   to display all variables, do not specify a name

   to display specific variables, specify names

   to unset specific variable, specify **-u** or **name=**

   options:

   -? or **-h** print a help message

   -d print diagnostics

   -u unset the specified variable

   Values that contain spaces must be quoted. More than one value can be set on the command line. The names of environment variables must start with a letter, number or underscore, and cannot contain the character '='.

   For example:

   ```
   set
   set A = 1 B = 2
   set C="a string value"
   set A B C
   set -u _EXIT_CODE
   set _0=
   ```

3. A new Catalyst **if** command has been added. This command is supported on the Propeller 2 only. It allows testing the value of environment variables and

performing actions based on the result. It is primarily intended to be used in Catalyst scripts.

The syntax of the command is:

if <name> [<action> [<param>]]

or

if <name> <unary_op> [<action> [<param>]]

or

if <name> <binary_op> <value> [<action> [<param>]]

where:

<name> is the name of an environment variable. If only a name is specified, the script exits if the variable does not exist

<unary_op> can be **exists** or **missing** (if no operator is specified, **exists** is assumed).

<binary_op> can be **equals**, **=**, **==**, **!=**, **<>**, **~=**, **<**, **<=**, **=<**, **>**, **>=**. **=>** (note Lua promotes string values to integers if used in an arithmetic expression - so the string "1" is the same as 1)

<value> can be a string or an integer (note that strings that contain spaces must be quoted).

<action> can be **continue**, **exit**, **skip**, **echo**, **add**, **sub**, **set**, **assign** or **prompt** (if no action is specified, **continue** is assumed).

<param> for **skip** this is the number of lines to skip when true (if no parameter is specified then 1 is assumed).

for **echo** this is a value to echo (if no parameter is specified then the value of the variable is echoed).

for **add** and **sub** this is the amount to add or subtract from the variable (a variable that does not exist or which has a value that cannot be interpreted as an integer has value 0).

for **set** this is the value to set in the variable (if no value is specified the variable is deleted).

for **assign** it is the name of another environment variable to assign to the variable.

for **prompt** it is the string prompt to display (if a value is entered it is stored in the variable, otherwise the variable is deleted). If no string is specified the name of the environment variable is used.

For example:

```
if A                    -- exits if variable A doesn't exist
if A continue           --  "
if A exists             --  "
```

```
if A missing exit       --  "
if A missing            -- exits if variable A exists
if A exists exit        --   "
if A skip               -- skips one line if variable A exists
if A missing skip 2     -- skips two line if variable A doesn't exist
if A = 1                -- exits if value of variable A is not 1
if A = 1 exit           -- exits if value of A is 1
if A = "2" skip 3       -- skips three lines if value of A is 2
if A != "a string" exit -- exits if value of A is not "a string"
if A echo               -- echo the value of A if A exists
if A missing echo "No!" -- echo "No" if A does not exist
if A > 0 sub 1          -- subtract 1 to A if value of A is > 0
if A add 1              -- add 1 to A if it exists
if A missing prompt ">" -- prompt for value for A with ">" if A missing
if A exists prompt      -- prompt for value for A with "A=" if A exists
if A missing set "ok"   -- set A to "ok" if A does not exist
if A set                -- set A to null (i.e. unset it) if A exists
if A missing assign _1  -- set A to the value of _1 if A is missing
```

Note that at least one space must separate each argument, so:

```
if A = 1                -- will continue if A equals 1
if A=1                  -- will generate an error and exit
if A= 1                 --  "
if A =1                 –  "
```

Note that case is significant in string values but not in the variable names, operations or actions.

Note that although only one condition can be specified in each "if" command, logical AND or OR of multiple conditions can be achieved using multiple "if" commands and the "skip" action.

For instance to exit if (A = 1) OR (B = 2) OR (C = 3):

```
if A = 1 exit
if B = 2 exit
if C = 3 exit
```

To exit if (A = 1) AND (B = 2) AND (C = 3):

```
if A != 1 skip 2
if B != 2 skip 1
if C == 3 exit
```


Note that loops can be created using the **if** command with the **skip** and **add** or **sub** actions. For instance, put the following in a file called *loop* (this file is provided as part of the pre-build demos):

```
@# use argument 1 if provided, otherwise prompt
@if _LOOP missing assign _1
@if _LOOP missing prompt "Number of times to execute? "
@# do loopy stuff here …
@if _LOOP echo
@# ... end loopy stuff
@if _LOOP sub 1
@if _LOOP = 0 skip 2
@# re-execute this script to loop (note use of "_0"!)
@exec _0
@if _LOOP > 0 skip 2
@# clean up after ourselves
@set _LOOP=
```

Then execute the script using the **exec** command. For example:

```
exec loop
```

or

```
exec loop 10
```

See the description of the **exec** command (below) for more details.

4. A new Catalyst **exec** command has been added. This command is supported on the Propeller 2 only. It allows executing a Catalyst script from the command line or from a Catalyst script and passing parameters to it.

   The format of the command is:

   exec script [parameters ...]

   The **exec** command works by passing the parameters to the script using environment variables. The Catalyst script name is passed in **_0**, and up to 22 parameters are passed in **_1** to **_22**.

   For example, the following command executes the Catalyst script in the file *script* and passes it the value **script** in variable **_0**, the value 3 in variable **_1** and the value "my string" in variable **_2**:

   ```
   exec script 3 "my string"
   ```

   Note that the "exec" command does PARAMETER SUBSTITUTION - any arguments passed to it of the form **_0** .. **_22** are substituted with the current value of the relevant environment variable. Note this is done BEFORE the new values are set in these variables. So an example of using exec in a Catalysts script might be:

   ```
   exec _0 _1 "a different string"
   ```

   This would re-execute the currently executing script (whose name is in variable **_0**), passing it the same first parameter as it was passed, but passing "a different string" in place of the second parameter.

   Note that the exec command passes control to the new script - i.e. it is not a **call** statement, has no stack, and never returns.

For a working example, see the loop example in the description of the new **if** statement.

5. The Catalyst **time** command has been updated to know about time zones. Previously the time displayed was always the time in the Real-time clock, and the time zone (as identified in the **TZ** environment variable) was hardcoded to "GMT", which meant no offsets were applied to that time. This can still be done if (for example) the time set in the RTC is the correct local time. However, now the RTC can be set to the correct UTC time and the **TZ** environment variable can be used to adjust this to the correct local time. The format and use of the **TZ** environment variable is described in:

   https://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html

   To set the time to one of the time zones hardcoded in the library file *source/lib/time/misc.c* just specify the 3 or more character time zone name.

   For example:

   ```
   set TZ=AEST
   ```

   To set the time zone to any time zone but WITHOUT corrections for daylight savings, give the time zone name and the offset required to convert that time zone to UTC. For example:

   ```
   set TZ=AEST-10
   ```

   or

   ```
   set TZ=AEST-10:00
   ```

   To set the time zone to any time zone WITH corrections for daylight savings, specify a fully qualified TZ. For example:

   ```
   set TZ=AEST-10AEDT,M10.1.0/2,M4.1.0/2
   ```

   The **time** command with no options, now prints local time in unix format. New options have been added to the **time** command to display the UTC time in either the 12 or 24 hour format used by the Parallax RTC driver code.

   time [options] [DD/MM/YY] [HH:MM:SS [AM | PM]]

   options:

   -? or **-h** print this helpful message and exit

   -b mode set battery mode (0 or 1) and report

   -c calibrate on RTC start

   -r software reset the RTC

   -d prompt for new date

   -t prompt for new time

   -u print UTC time in 12 hour format

   -w print UTC time in 24 hour format

   -v verbose mode

   The **time** command is supported on the Propeller 2 only.

6. The precompiled demo versions of Catalyst for the P2_EVAL and P2_EDGE (i.e. *P2_EVAL.ZIP, P2_EVAL_VGA.ZIP*, *P2_EDGE.ZIP* and *P2_EDGE_VGA.ZIP*) now all assume a Parallax RTC add-on card is installed on pin 24. However, if one is not, then apart from the **time** command, which will report that an RTC is not detected, everything should work correctly.

   Note that the pre-build version in *P2_DEMO.ZIP* does NOT assume an RTC is present, and still uses only the software clock.

7. The self-hosted version of Catalina (supported on the Propeller 2 only) now uses environment variables, such as **CATALINA_DEFINE**, just as it does on Windows or Linux. This effectively removes the limitations of Catalyst only allowing 24 command line arguments. If the catalina command detects that this would happen, it now prints a message suggesting the use of the **CATALINA_DEFINE** environment variable instead of passing parameters on the command line.

   The pre-built versions of Catalyst now set the necessary environment variables to the correct defaults.

   Catalyst now includes a new **cat_env** command that does the same job as the **catalyst_env** command does on Windows or Linux. For example, in the case of the pre-built version in *P2_EDGE.ZIP* it would display:

```
CATALINA_DEFINE   = P2_EDGE SIMPLE VT100 CR_ON_LF USE_COLOR
CATALINA_INCLUDE  = /include
CATALINA_LIBRARY  = /target
CATALINA_TARGET   = /lib
CATALINA_TEMPDIR  = /tmp
LCCDIR            = [Default]
```

8. The Catalina sub-processes in the self-hosted version of Catalina (i.e. **cpp**, **rcc**, **bcc**, **spp**, **pstrip**, **p2asm**) now set the environment variable **_EXEC_CODE** to **0** on success, or **1** on failure. This allows errors in the compilation to be detected and the compilation process to be terminated without waiting for it to complete (which can take a long time!).

   To do this, the **catalina** command now adds the following line to the script it generates that invokes each sub-process:

```
if _EXIT_CODE != 0 exit
```

   Refer to the description of the new **if** command for details.

Other Changes

6. A problem in the **`_register_plugin()`** and **`_unregister_plugin()`** library functions which could result in the error message that the symbol **`C__registry`** was not defined has been fixed. Affected the Propeller 1 and Propeller 2.

7. The library file *misc_ytab.c* had a name incompatible with the DOS 8.3 character file names, and has been renamed to *misc_yt.c*. Affected the self-hosted version of Catalina on the Propeller 2 only.

8. The Lua functions **`propeller.getpin()`** and **`propeller.setpin()`** were not validating their arguments correctly. Affected the Propeller 1 and Propeller 2.

9. An error in the RTC driver **`rtc_time_24()`** function has been fixed. Affected the Propeller 2 only.

Release 6.3

New Functionality

1. A new Catalina HMI option has been implemented on the Propeller 1, which can be selected by defining the Catalina symbol **TTY256** (e.g. by including **-C TTY556** on the Catalina command line). This HMI option is applicable to all supported Propeller 1 platforms. It can be specified wherever **TTY** could be specified. It tells Catalina to use the serial interface plugin with 256 byte transmit and receive buffers instead of the normal one, which uses 16 byte buffers. Note that using this option instead of **TTY** can improve serial performance and reduce the chance of lost bytes, but it also increases program sizes by around 500 bytes.

2. Catalina now supports the Parallax Real-Time Clock module on the Propeller 2. This support is implemented in the existing software clock plugin, and is enabled by specifying the Catalina symbol **RTC** instead of **CLOCK**. Note that if **RTC** is specified, the clock plugin will always use a separate cog, whereas previously if both the clock and the SD card plugin were in use they could share the same cog. This means that programs that use the RTC hardware clock may require one more cog than programs that use only the software clock.

   While the clock plugin accepts a new time being set, the new time is still only set in the plugin - i.e. it only applies until the Propeller is reset. To actually change the RTC hardware time, separate software must be used. This is both for backwards compatibility, and because the time setting software is too large to fit in the clock plugin. Instead, stand-alone RTC/I2C drivers (in files *demos\catalyst\time\rtc_driver.c* and *demos\catalyst\time\i2c_driver.c*) are provided to do this, and a Catalyst utility program called **time** is also provided. This utility allows setting the RTC hardware time as well as various associated RTC parameters from the Catalyst command line. A C version of the Parallax RTC example program is also provided (in file *demos\catalyst\time\rtc_example.c*).

3. Catalina's stdio file functions now set the file creation and modification date and time if either the RTC hardware clock is in use, or the software clock is in use and the time has been set (to the year 2000 or later). If not, the default DOSFS time of 1/1/2006 01:01:00 is used. Note that the file access date is never modified after file creation, because modifying this on every file access would slow down the file system too much.

4. Catalyst's self-hosted version of Catalina now prints the time the compilation starts and finishes if very verbose mode is specified (by adding the command line options **-v -v**).

5. The inter-character delay introduced in the payload terminal emulator in version 6.2 was set at a fixed 100 ms, but this value caused problems in some cases, so it has been reduced to 25 ms by default, and a new command-line option (-K) has been introduced to set it.

   A better solution for Propeller 1 programs that have trouble keeping up with the shorter inter-character delay is to use the **TTY256** HMI option instead of the **TTY** HMI option.

Other Changes

1. The Catalyst ls utility was not printing the file times correctly when the very long output option was specified (by adding the command line options **-ll** or **-l -l**) . Affected both the Propeller 1 and 2.

2. The stdio file system functions were not setting the file dates correctly when creating files. Affected both the Propeller 1 and 2.

3. When building Catalyst on the Propeller 1, there are more cases when the option **EEPROM_CATALYST** will need to be included. such as if the new **TTY256** HMI option is used. See the README.TXT file in the demos\catalyst folder for more details.\

4. The **_pinclear()** function was not working correctly. Affected the Propeller 2 only.

5. When the VGA HMI option was in use, the self-hosted version of Catalina did not have enough cogs to run the **rcc** component. Also, the catalina command would prompt to continue at several points during the compilation. Affected the Propeller 2 only.

Release 6.2

New Functionality

6. The lua execution engine can now be built to use either PSRAM or HYPER RAM for code - just as you could add **ENABLE_PRSAM** to to the normal Catalyst or Lua build_all script, you can now add **ENABLE_HYPER**. The resulting executable will be called *src/luaxp.bin*. Refer to the Catalyst Reference Manual, and the section in it called Propeller 2 Platform-specific Notes for more details.

7. Two new Multi-processing demos have been added - diners.lua models the classic "dining philosopher" problem, and solved.lua offers one possible solution to it.

8. All versions of Lua now accept a new optional parameter to the version function in the propeller module - the string "hardware". For example:

   ```
   propeller.version("hardware")
   ```

   will now return:

   0 on Windows or Linux

   1 on a Propeller 1

2 on a Propeller 2

9. A new SPI demo program (*demos\spi\dump_eeprom.c*) has been added. See the demos\spi folder for details.

10. New PASM test programs have been added to test the PASM _PSTR macro. See the *demos\pasm_pstr* folder for details.

11. A new demo program has been added (*demos\examples\ex_str_print.c*) to print values to strings.

Other Changes

6. The Lua Windows Makefile has been updated to build the Multi-processor version of Lua (i.e. **mlua** and **mluax**) correctly for Windows using mingw, or Linux using gcc. Note that the Propeller version was being built correctly - it uses a different Makefile (*Makefile.Catalina*).

7. The Lua **build_all** script and Catalina Makefile (*Makefile.Catalina*) now accept any memory model on the command line, whereas previously they would only work for **SMALL** and **LARGE**. If no memory model is specified they still default to **LARGE** for the Propeller 1 or **COMPACT** for the Propeller 2, but it is now possible to override this (for instance) to build Lua in **NATIVE** mode on the Propeller 2, which is much faster than **COMPACT** mode for small Lua programs.

8. The Windows and Linux versions of Multi-processing Lua (**mlua**) and the corresponding Multi-processing Lua Execution Engine (**mluax**) have been updated to accept requests for more than one factory without generating an error message. This was for consistency with the Propeller versions, which already did so, and requests for more factories than there were available cogs would simply result in all available cogs being used. However, on both Windows and Linux only one factory will actually be used.

9. Geany has been updated to version 1.37.1. This was primarily a bug fix release, with no significant functional changes.

10. Details build instructions have been added for Windows and Linux - see the file *BUILD.TXT*.

11. Some Windows components now need to be built using Cygwin, and the remainder are now built with the current version of MINGW and MSYS2. The build scripts have been updated (see the file *BUILD.TXT* for more details) and the installer now installs the necessary Cygwin DLL (as b*in\cygwin1.dll*).

12. Various bugs have been fixed in the following scripts. Affected Linux only:

    build_utilities
    catalina_geany
    Set_Linux_Permissions
    flash_payload

13. The maximum number of symbols the Spinnaker (OpenSpin) compiler can have in its symbol table has been increased from 8192 to 16384 to allow the compilation of very large programs. In particular, the Catalyst "vi" text editor

would not compile if the Catalina Optimizer was used, which causes the generation of more symbols. Affected only the Propeller 1.

14. Details on the licensing of Catalina and programs built with Catalina have been clarified and updated. There have been no changes to the actual license conditions which are essentially GPL for Catalina itself and Lesser GPL (aka LGPL) for programs built USING Catalina (see the files *COPYING* and *COPYING.LESSER* in the Catalina directory, and also the file *Copyright* in the *sources/lib* directory for the C89 library, which is based largely on the Amsterdam Compiler Kit). The only time you have to impose additional restrictions on a program BUILT with Catalina is if you distribute the Catalina Target Package or C89 library (e.g. because you have modified them and want to be able to distribute sources to your program). In that case, you must also distribute the modified components according to the terms of the appropriate license.

15. On Windows, payload now uses ncurses rather than pdcurses by default. This means that the window size cannot be set from within the program. The -g command line option sets the terminal geometry parameters, but no longer resizes the terminal window, which must be done manually. This was always the case on Linux, so this change makes both platforms now behave the same way. To ease the transition, a new batch file (*payloadi* on Unix, *payloadi.bat* on Windows) is included with Catalina to do this for you when the -g parameter is specified. Use **payloadi** just as you would use **payload**.

16. A delay between characters has been added to payload's interactive terminal emulator, since the Propeller 1 TTY interface has such a small character buffer (16 chars) it is very easy to overwhelm it by holding a key down. This was a particular problem for programs such as the vi editor when the cursor movement keys are held down.

Release 6.1.1

New Functionality

1. A **_PSTR()** macro has been defined, which can be used within calls to the **PASM()** function to interpret C escape sequences in strings. See the Catalina Reference Manual for details.

Other Changes

None.

Release 6.1

New Functionality

1. The inline PASM capabilities have been enhanced by the addition of a new **_PASM()** macro, which simplifies the use of C identifiers in inline PASM strings used as arguments to the **PASM()** function. New demo programs have been added to *demos\inline_pasm* to illustrate this, and all the existing demo programs have been updated as well. Also, all the inline PASM demo programs now work "as is" on the **C3**, **HYDRA**, **P2_EVAL** and **P2_EDGE**

platforms. The Catalina Reference Manual has also been updated with more information about inline PASM.

2. The **build_utilities** script now has the option of copying the utilities it builds to the current directory as well as to Catalina's *bin* directory. If neither of these options is selected, the utilities are built and left only in the *utilities* directory itself. Recall that the script will use the user's utilities folder if one exists in the user's home directory, otherwise it will use the one specified by the directory in the **LCCDIR** environment variable.

3. The accuracy of the following timer functions has been improved:

   _waitus()
   _waitms()
   _waitsec()
   _iwaitus()
   _iwaitms()
   _iwaitsec()

   A new demo program has been added to the *demos\examples* folder - the *ex_timers.c* program will show the accuracy of all the timer functions.

   Note that the these functions are most accurate for programs executed entirely from Hub RAM (i.e. **TINY**, **COMPACT** or **NATIVE** programs) and when neither multi-threading nor interrupts are used. They are less accurate when used in multi-threaded, interrupt-driven, or **SMALL** or **LARGE** XMM programs, especially when the cache has to be used and the cache size is small. Catalina's multi-model support can be used to place time-critical sections of code entirely in Hub RAM and use dedicated cogs. Other parts of the programs can be executed from XMM RAM.

4. The Comms VT100 terminal emulator has a new option added to the **Edit** menu - **Clear Buffer**, which clears the entire buffer and resets the cursor, screen and view to the beginning of the buffer.

Other Changes

1. A bug that may have caused the **_waitus()**, **_waitms()** and **_waitsec()** timer functions to hang for up to 53 seconds has been fixed. Affected only the Propeller 1.

2. Some programs built in **COMPACT** mode were not working correctly, such as *demos\graphics\graphics_demo.c* - this was due to an error in the c*atalina_compact.inc* file. Affected only the Propeller 1.

3. The Catalina blackbox debugger was not working on programs in **COMPACT** mode due to an error in the file *catalina_compact.inc*. Affected only the Propeller 1.

4. The file *pthread.h* should have included the file *rtc.h* in order to correctly define the **rtc_settime()** function. Affected both the Propeller 1 and 2.

5. The *examples**ex_leds_2.c* demo program was using the wrong value for a LED on the **P2_EDGE** board. Affected only the Propeller 2.

6. Various demo programs have been tidied up. Many minor issues have been corrected. More significant issues have been addressed in the following demo programs:

   - *demos/games/chimaera.c* was not being compiled with the extended C library, so the logging function was not working. Affected both the Propeller 1 and 2.

   - *demos/benchmark/ackerman_2.c* still had a call to **printf()**, which meant that instead of being smaller than *ackerman_1.c*, it ended up larger. Affected both the Propeller 1 and 2.

   - *demos/spinc/test_pasm.c* now builds for both **TINY** and **COMPACT** mode. Also, the *Makefile* now warns correctly if these programs are built for a Propeller 2 (**spinc** is specific to Propeller 1).

7. The *Makefile* in the *demos**examples* folder knows more about the options required to build particular programs, and which programs apply only to the Propeller 1 or 2, which means less error messages are now generated when the **build_all** batch script is used. Affected both the Propeller 1 and 2.

Release 6.0.1 Errata

New Functionality

1. A default **clean_all** script has been added to the *bin* directory. If the current directory does not have a **clean_all** script, this provides a default one that will delete any Catalina binaries, listings, and also some temporary files from the current directory.

2. The *catalina.lua* script (used by the self-hosted version of Catalina) has been updated to print a help message when it is invoked with no parameters.

Other Changes

1. Fixed some errors in the 6.0.1 README.TXT file.

2. Updated the Catalyst reference manual to include details about aborting auto-execution scripts, and maximum program sizes.

3. Modified some demo programs to use Unix line terminators, not DOS - this makes them easier to view and edit in the vi text editor.

Release 6.0.1

New Functionality

1. Catalina now has the option of caching SD card sector reads and writes when PSRAM or HYPER RAM is available. The cache supports two modes:

   Write Through if the sector is in the cache then it is used for reads, but all writes are done both to the cache and to the SD card. This is the default mode.

Write Back if the sector is in the cache then it is used for reads, and all writes are done only to the cache, with the sector in the cache marked as dirty. To write all the dirty sectors in the cache back to the SD card, an explicit cache flush must be performed. It is fine not to flush the cache if the cache has not been filled, but once it has been filled then any sectors not cached will be written directly to the SD card, and so if the cache is NOT subsequently flushed then the SD card can end up in a corrupt or inconsistent state.

The cache is supported only on the P2_EDGE and P2_EVAL boards, and is enabled by compiling the library with the PSRAM or HYPER option specified and then linking the programs that use it with both the extended C library and the appropriate PSRAM or HYPER library (i.e. **-lcx** or **-lcix** and **-lpsram** or **-lhyper**). Since all programs linked with the library built to use the cache will HAVE to be linked this way, it is recommended that the general version of the library NOT enable the cache, but that a separate version of the library be compiled to use with this option. For an example, see the **build_psram** and **build_hyper** scripts in the *source/lib* folder, and how these are used by the **build_all** script in *demos/catalyst/catalina*

The P2_EVAL board with onboard PSRAM has 32MB of PSRAM, so Catalina uses the lower 16MB as XMM RAM, and the upper 16MB as SD card cache. Catalina can only use 16MB of XMM RAM, but if some of the PSRAM is required for other purposes, the size of the SD card cache can be adjusted - *see source/lib/include/sd_cache.h*.

The HyperFlash/HyperRAM add-on board has 16MB of PSRAM, so by default Catalina uses the lower 14MB as XMM RAM, and the upper 2MB as SD card cache. The amount used for the SD card cache can be adjusted - see *source/lib/include/sd_cache.h*. Note that no checking is done on the use of XMM RAM - so if the program uses XMM RAM above the 14MB (e.g. if it allocates too much heap space), then it will overwrite the cache. In that case, it is better to not use the cache at all.

The SD card cache is currently only used by the self-hosted version of Catalina. When compiled as a Catalyst demo, the following catalina-related programs use the cache (in Write Back mode) to significantly improve SD card performance:

cpp
rcc
bcc
spp
pstrip
p2asm

Using the cache can reduce the time required for the self-hosted version of Catalina to compile a C program by 20% to 33%.

For details of the functions supported by the cache, see the file *source\lib\include\sd_cache.h*

Other Changes

1. The reduction in SD card driver delay time made in release 5.9.2 has been reverted because it did not work reliably on all SD cards, some of which appear to require the longer delay. However, this change only made a small improvement in very SD card intensive programs (such as catalina itself) when compared to the new sd card cache option.

Release 6.0

New Functionality

1. There is a new Catalyst demo - Catalina itself! The *demos\catalyst* folder has a new subdirectory called *catalina*, in which a self-hosting version of Catalina can be built. Since it is supported only on a limited number of Propeller 2 platforms, this new demo is not built by default when you build Catalyst. If you have a P2 EDGE board with PSRAM or a P2 EVAL board with HyperRAM then you can build it manually - go to the *catalina* subdirectory, and use the **build_all** script with the same options you used to build Catalyst. For example

   ```
   cd catalina
   build_all P2_EDGE SIMPLE VT100 CR_ON_LF USE_COLOR OPTIMIZE MHZ_200
   ```

   or

   ```
   cd catalina
   build_all P2_EVAL VGA COLOR_4 OPTIMIZE MHZ_200
   ```

   Note that the **build_all script** will automatically use the **copy_all** script to copy the results to the Catalyst *image* folder.

   The self-hosted version of Catalina currently has the following limitations:

   - it supports the Propeller 2 only.

   - it supports the **TINY**, **COMPACT** and **NATIVE** modes only.

   - it does not support the Catalina debugger, optimizer or parallelizer.

   Since the self-hosting version of catalina introduces a *bin* directory to hold the catalina executables, the **build_all** and **copy_all** scripts now put their output in a directory called *image* instead of *bin*. See the Catalyst Reference Manual for more details on the self-hosted version of Catalina.

2. Local user libraries must now be created in a local subdirectory called *lib*. For example, if you specify **-liberty** on the command line, previous versions of Catalina would first look for a local library in a subdirectory in the current directory called *libiberty*, but now Catalina will look for a it in a local subdirectory called *lib\iberty*.

3. A new Catalina symbol **P2_REV_A** can now be defined (e.g. using **-C P2_REV_A** to the catalina command) to indicate that p2asm should not generate instructions not supported by the Rev A version of the Propeller 2. Previously, the **p2_asm** batch script had to be edited to do this. Note that this applies only to the Catalina compiler itself, which no longer uses the **p2_asm** script. The **p2_asm** script must still be manually edited (to remove the -**v33** option to **p2asm**) if the Catalina Optimizer is used.

Other Changes

1. There have been no changes to functionality, but the spin preprocessor has been renamed from **spinpp** to **spp**. This better reflects that this program is mainly used not to preprocess Spin source files, but to preprocess assembly language source files, which by convention are given a **.s** extension.

2. There have been no changes to functionality, but the C library has been extensively modified in both file names and structure to allow it to be used on systems that support only DOS 8.3 file names. This includes changing the names of various propeller include files, as follows:

```
catalina_cog.h    ==>  cog.h
catalina_commondrv.h    ==>  commdrv.h
catalina_float.h  ==>  floatext.h
catalina_fs.h     ==>  fs.h
catalina_gamepad.h==>  gamepad.h
catalina_graphics.h     ==>  graphics.h
catalina_hmi.h    ==>  hmi.h
catalina_hyper.h  ==>  hyper.h
catalina_icc.h    ==>  caticc.h
catalina_interrupts.h   ==>  int.h
catalina_plugin.h ==>  plugin.h
catalina_psram.h  ==>  psram.h
catalina_rtc.h    ==>  rtc.h
catalina_sd.h     ==>  sd.h
catalina_serial2.h==>  serial2.h
catalina_serial4.h==>  serial4.h
catalina_serial8.h==>  serial8.h
catalina_sound.h  ==>  sound.h
catalina_spi.h    ==>  spi.h
catalina_threads.h==>  threads.h
catalina_tty.h    ==>  tty.h
catalina_vgraphics.h    ==>  vgraphic.h
mathconst.h       ==>  mathcnst.h
propeller.h       ==>  prop.h
propeller2.h      ==>  prop2.h
smartpins.h       ==>  smartpin.h
thread_utilities.h==>  thutil.h
```

   To ease the transition, the existing include files have been retained for systems that support long file names, including Windows and Linux, so on those platforms you can either use the old names or the new names. However, it is recommended that new programs use the new file names so that they can be compiled on any system that supports Catalina.

3. There have been no changes to functionality, but the target files have been extensively modified in both file names and structure to allow them to be used on systems that support only DOS 8.3 file names. This includes putting the Propeller 1 specific target files in a subdirectory called p1, and the Propeller 2 specific target files in a subdirectory called p2, and also changing the names of various target files, as follows:

```
Catalina_reserved_null.inc          => reserven.inc
Catalina_reserved.inc               => reserve.inc
Catalina_defines.inc                => define.inc
Catalina_constants.inc              => constant.inc
Catalina_arguments.inc              => argument.inc
Catalina_platforms.inc              => platform.inc
```

```
Catalina_plugins.inc                => plugin.inc
Catalina_threading.inc              => thread.inc
Catalina_blackcat.inc               => blackcat.inc
catalina_compact.inc                => compact.inc
debug_led.inc                       => debugled.inc

catalina_default.s                  => def.t
catalina_blackcat.s                 => dbg.t

P2_CUSTOM.inc                       => P2CUSTOM.inc
P2_EVAL.inc                         => P2EVAL.inc
P2_EDGE.inc                         => P2EDGE.inc

PSRAM_XMM.def                       => psram.def
PSRAM_XMM.inc                       => psram.inc
HYPER_XMM.inc                       => hyper.inc
Cached_XMM.inc                      => cached.inc

Catalina_kernel_library.inc         => klib.inc
Catalina_thread_library.inc         => thlib.inc
Catalina_pre_sbrk.inc               => presbrk.inc

Catalina_CMM_kernel_library.inc     => cmmklib.inc
Catalina_LMM_kernel_library.inc     => lmmklib.inc
Catalina_NMM_kernel_library.inc     => nmmklib.inc

unscii-16.inc                       => font16.inc
unscii-8.inc                        => font8.inc
unscii-8-fantasy.inc                => font8f.inc
unscii-8-thin.inc                   => font8t.inc

BlackCat_DebugCog.spin2             => debugcog.t
Cache.spin2                         => cache.t

P8X32A_ROM_TABLES.spin2             => p1rom.t
Flash_loader_1.2_mod2.spin2         => flshload.t

Catalina_Cache.spin2                => cogcache.t
Catalina_CogStore.spin2             => cogstore.t
cogserial.pasm                      => cogs2.t
hyperdrv.spin2                      => coghyper.t
psram16drv.spin2                    => cogpsram.t
MultiPortSerial.pasm                => cogs8.t
Catalina_SD_Plugin.spin2            => cogsd.t
Catalina_vga_tile_driver.spin2      => cogvga.t
Catalina_1CogKbM_A.spin2            => cogkbma.t
Catalina_1CogKbM_B.spin2            => cogkbmb.t
Catalina_1CogKbM_Common.spin2       => cogkbmc.t
Catalina_1CogKbM_pre_sbrk_A.spin2   => kbmprea.t
Catalina_1CogKbM_pre_sbrk_B.spin2   => kbmpreb.t

Catalina_Float32_A_Plugin.spin2     => floata.t
Catalina_Float32_B_Plugin.spin2     => floatb.t
Catalina_Float32_C_Plugin.spin2     => floatc.t

Catalina_HMI_Plugin_SIMPLE.spin2    => hmisimpl.t
Catalina_HMI_Plugin_TTY.spin2       => hmitty.t
Catalina_HMI_Plugin_VGA.spin2       => hmivga.t

Serial2.spin2                       => serial2.t
Serial8.spin2                       => serial8.t
```

```
Catalina_RTC_Plugin.spin2           => cogrtc.t
Clock.spin2                         => clock.t
SDCard.spin2                        => sd.t
PSRAM.spin2                         => psram.t
HYPER.spin2                         => hyper.t
Float.spin2                         => float.t
HMI.spin2                           => hmi.t

Catalina_HUB_XMM_Loader.spin2       => hubload.t
Catalina_SD_Loader.spin2            => sdload.t

Catalina_CMM_library.spin2          => cmmlib.t
Catalina_LMM_library.spin2          => lmmlib.t
Catalina_NMM_library.spin2          => nmmlib.t
Catalina_XMM_library.spin2          => xmmlib.t

cmm_progbeg.s                       => cmmbeg.t
cmm_progend.s                       => cmmend.t
cmm_default.spin2                   => cmmdef.t
cmm_blackcat.spin2                  => cmmdbg.t

emm_progbeg.s                       => emmbeg.t
emm_progend.s                       => emmend.t
emm_default.spin2                   => emmdef.t
emm_blackcat.spin2                  => emmdbg.t

smm_progbeg.s                       => smmbeg.t
smm_progend.s                       => smmend.t
smm_default.spin2                   => smmdef.t
smm_blackcat.spin2                  => smmdbg.t

lmm_progbeg.s                       => lmmbeg.t
lmm_progend.s                       => lmmend.t
lmm_default.spin2                   => lmmdef.t
lmm_blackcat.spin2                  => lmmdbg.t

nmm_progbeg.s                       => nmmbeg.t
nmm_progend.s                       => nmmend.t
nmm_default.spin2                   => nmmdef.t
nmm_blackcat.spin2                  => nmmdbg.t

xmm_progbeg.s                       => xmmbeg.t
xmm_progend.s                       => xmmend.t
xmm_default.spin2                   => xmmdef.t
xmm_blackcat.spin2                  => xmmdbg.t

Catalina_CMM.spin2                  => cmm.t
Catalina_CMM_dynamic.spin2          => cmmd.t
Catalina_CMM_threaded.spin2         => cmmt.t
Catalina_CMM_threaded_dynamic.spin2 => cmmtd.t

Catalina_LMM.spin2                  => lmm.t
Catalina_LMM_dynamic.spin2          => lmmd.t
Catalina_LMM_threaded.spin2         => lmmt.t
Catalina_LMM_threaded_dynamic.spin2 => lmmtd.t

Catalina_NMM.spin2                  => nmm.t
Catalina_NMM_dynamic.spin2          => nmmd.t
Catalina_NMM_threaded.spin2         => nmmt.t
Catalina_NMM_threaded_dynamic.spin2 => nmmtd.t
```

```
Catalina_XMM.spin2                  => xmm.t
Catalina_XMM_dynamic.spin2          => xmmd.t
```

Note that the extension **.t** is used instead of **.s** for assembly language source files in the target directory. This prevents name collisions with user programs.

4. There have been no changes to functionality, but the way Catalina names and structures its system libraries has been modified to accommodate systems that support only DOS 8.3 file names. The changes are as follows:

```
compact_lib       => lib\p1\cmm
compact_lib_p2    => lib\p2\cmm
large_lib         => lib\p1\xmm
large_lib_p2      => lib\p2\xmm
lib               => lib\p1\lmm
lib_p2            => lib\p2\lmm
native_lib_p2     => lib\p2\nmm
```

So, for example, when you use an option like **-lthreads** on the command line to use the threads library when compiling an **XMM LARGE** program for a Propeller 1, previous versions of Catalina would expect the library to be in directory *large_lib\libthreads*, but it now expects the same library to be in *lib\p1\xmm\threads*.

5. There have been no changes to functionality, but the catbind program now uses the default name *catalina.idx* for library indexes (instead of catalina.index) to accommodate systems that support only DOS 8.3 file names. Any existing catalina.index files can simply be renamed, or deleted and regenerated using **catbind** or **bcc**.

6. There have been no changes to functionality, but Catalina now uses the default target directory of *target\p1* for the Propeller 1, or *target\p2* for the Propeller 2, to accommodate systems that support only DOS 8.3 file names.

7. There have been no changes to functionality, but Catalina now uses the name *vgraphic* for the virtual graphics library instead of *vgraphics*, to accommodate systems that support only DOS 8.3 file names. So programs that used to use the command-line option **-lvgraphics** should now use **-lvgraphic** instead.

8. There have been no changes to functionality, but to accommodate platforms that support only DOS 8.3 file names, Catalina now looks in a subdirectory of the target directory to find the target files based on the version of the Propeller. For the Propeller 1 it expects the files to be in a subdirectory called *p1* and for the Propeller 2 it expects them to be in a subdirectory called *p2*. On the catalina and catbind command lines, the target is still specified without the version, so to use the *minimal* target, you still just use a command like:

```
catalina hello_world.c -T "%LCCDIR%\minimal"
```

Release 5.9.3

New Functionality

1. Catalyst auto-execution is now slightly more sophisticated. First, the *AUTOEXEC.TXT* file can also now contain a script consisting of multiple commands - however, unlike the *EXECONCE.TXT* file, when the last

command in the *AUTOEXEC.TXT* file has been executed, the entire file will then be re-executed.

Also, just before it executes each command, Catalyst now checks for a key press and offers to abort the command auto-execution if one is detected. Previously, there was no way to abort the execution of a sequence of commands once initiated other than removing the SD card. Note that since the key buffer is reset on each reboot and so the key will only be detected between the reboot and the execution of the next command, it will usually be necessary to hold a key down until the current command completes. To abort the current command, hold a key down and reset the Propeller.

Also, Catalyst now echoes commands in the *AUTOEXEC.TXT* or *EXECONCE.TXT* files before execution unless the first character of the command is '@' (which is ignored). Also, if the first character in the command line is '#' then the line is treated as a comment. Note that this functionality does not interfere with the use of the '!', '@' and '#' characters to specify the CPU if multi-cpu support is enabled, since that only applies to commands entered interactively.

Note that since it takes too much space on some Propeller 1 platforms (depending on the XMM and HMI options selected) the default is to disable command scripting. To enable scripting it may be necessary to disable something else to make enough space. For example, Lua commands could be disabled instead.

Note that disabling Lua commands does not disable Lua itself, it just disables the auto-detection and execution of Lua programs directly from the command line. Lua programs can always be executed by explicitly passing them to Lua. For example, if Lua commands are enabled, the lua program list.lua can be executed using a command like:

```
list *.bas
```

If Lua commands are disabled, the command required would be:

```
lua list.lua *.bas
```

2. The Catalyst **rm** command now has a **-k** option to kill (suppress) information messages.

Other Changes

1. Payload was incorrectly interpreting the VT100 escape sequence **ESC [2 J** as ‘erase to end of screen’, instead of ‘erase entire screen’. Affected both the Propeller 1 and 2.
2. Payload was incorrectly interpreting the VT100 cursor position escape sequence **ESC [r ; c H** when 0 was specified as the row or column position. It now accepts either 0 or 1 to mean the first position. Payload also now accepts **ESC [r ; c f** to mean the same thing. Affected both the Propeller 1 and 2.
3. The vi text editor was using the VT100 escape sequence **ESC c** to reset the terminal to the initial state when it started - but it is not clear what the initial state for some of the DEC private options should be, so it now simply clears the screen instead - this avoids resetting some VT100 options (such as Auto

Wrap) with some VT100 terminal emulators. Affected both the Propeller 1 and 2.

4. The *demos\p2_ram* examples would not compile correctly for the P2_EVAL board with the HyperRAM add-on board. Affected the Propeller 2 P2_EVAL only.
5. The *demos\interrupts* example programs now use the correct LEDs on the P2_EDGE boards (pins 38 & 39) and P2_EVAL boards (pins 56, 57 & 58).
6. The Lua execution engine (**luax**) was not being built correctly when the ENABLE_PSRAM option was specified.
7. The embedded target had not been updated with recent changes to the default target, such as the improved CLOCK plugin.

Release 5.9.2

New Functionality

1. None.

Other Changes

1. The changes in the round() and trunc() functions in Release 5.9 stopped the Catalyst pascal compiler from compiling correctly. Affected the Propeller 1 and Propeller 2.
2. The p2asm assembler has been modified to improve its speed. Thanks to Wuerfel_21.
3. The p2asm assembler is used on the Propeller 2 only.The SD card plugin had an error that could cause a failure to write a sector to go undetected. Also, some delays have been reduced to speed up the writing of sectors. Affected the Propeller 2 only.

Release 5.9.1

New Functionality

1. None.

Other Changes

1. Fixed a bug in the copying of components of structures in the XMM LARGE mode on the Propeller 2. Components were being copied as longs, which could fail if the start and end were not both aligned on long boundaries. Affected the Propeller 2 only - on the Propeller 1 all components of structures are copied as bytes.
2. The _cogstart_XMM functions added in release 5.6 have been removed from the C library and added as stand-alone functions where needed (e.g. see the folder *demos/p2_ram*).

   The functions affected are:

   ```
   _cogstart_XMM_SMALL();
   ```

```
_cogstart_XMM_SMALL_cog(g);
_cogstart_XMM_LARGE();
_cogstart_XMM_LARGE_cog();
_cogstart_XMM_SMALL_2();
_cogstart_XMM_SMALL_cog_2();
_cogstart_XMM_LARGE_2();
_cogstart_XMM_LARGE_cog_2();
```

When these functions were introduced Catalina supported only one type of XMM RAM on the Propeller 2, but now that multiple types of XMM RAM are supported (currently Catalina supports the internal PSRAM on the P2 Edge, and the HyperRAM add-on board on the P2 Edge or P2 Eval boards) these functions now need to be compiled specifically for the platform on which they are to be used, and so can no longer be included in the C library.

3. The Propeller 1 CUSTOM configuration did not have **ACTIVATE_EACH_USE_SD** defined in *Custom_CFG.inc*, which made the SD plugin too large to load.
4. Payload's simple-minded internal VT100 emulator now ignores color escape codes, so if vi is compiled with **USE_COLOR** defined (e.g. to work better with the external VT100 emulator) then using payload's internal terminal emulator will still work correctly (but note no color will be displayed). Previously, color escape codes were not recognized and were treated as output to be displayed on the terminal, which garbled the output, whereas now they are recognized and ignored.

Release 5.9

New Functionality

1. The multiport serial libraries (serial2, serial4, serial8) have each had two new functions added:

```
int sX_txcount(unsigned port)
int sX_rxcount(unsigned port)
```

where X = 2, 4 or 8. The functions return the current count of characters waiting in the transmit or receive buffers.

2. New variants of the maths functions to truncate and round floats have been added, called ltrunc() and lround(), which return longs. The original versions of **trunc()** and **round()** were incorrectly defined in the include file catalina_float.h as returning floats - in fact, they both returned long values. These functions have been retained, but have been renamed as **_round()** and **_trunc()**. These functions were only available when a maths co-processor library option was specified - e.g. if **-lma**, **-lmb** (P1 or P2) or **-lmc** (P2 only), and were not available in the software-only library option **-lm**.

   The new variants are called **lround()** and **ltrunc()** and return longs. They are available when any maths library option is used - i.e. including the **-lm** option. Also, new functions have been added called **fround()** and **ftrunc()** which return floats.

So now a program that needs **round()** or **trunc()** can choose which to use either by calling them directly, or by adding suitable #defines, such as

EITHER:

```
#define round(x) lround(x)
#define trunc(x) ltrunc(x)
```

OR:

```
#define round(x) fround(x)
#define trunc(x) ftrunc(x)
```

OR:

```
#define round(x) _round(x)
#define trunc(x) _trunc(x)
```

depending on whether it needs versions that return longs, versions that return floats (or doubles - in Catalina doubles are the same as floats, so if the program expects these functions to return doubles, it will work perfectly well with the new versions) or it wants to use the original versions.

3. On the Propeller 1, the functions **_waitsec()**, **_waitus()** and **_waitms()** use the **WAITCNT** instruction, and are "thread-safe", but on the Propeller 2 they use timer 2 and the **WAITCT2** instruction. This means they are not "thread-safe" since waiting on a timer also stalls interrupts. However, there are now "interrupt-safe" versions provided for the Propeller 2, called **_iwaitsec()**, **_iwaitms()** and **_iwaitus()**. These use the **POLLCT2** instruction and so do not stall interrupts, but they busy-wait. This means they consume more power, and they will also be very slightly less accurate, but accuracy cannot be guaranteed in a multi-threaded program, or one that uses interrupts.

Other Changes

1. The Catalina Propeller 2 Reference Manual said the 8 port serial functions **s8_openport()** and **s8_closeport()** returned an int. In fact, they both return void - i.e. they don't return anything. The manual has been updated.
2. The Catalina maths functions **round()** and **trunc()** were incorrectly defined in catalina_float.h as returning floats - in fact, they both returned long values. See the New Functionality section for more details.
3. Catalina's original implementation of trunc(), which was also used when a float was "cast" to an integer, was based on Cam Thompson's original Float32_A co-processor, which unnecessarily limited the range in which it would return a value. That range has been extended to include the maximum range that can be encoded accurately in a float. Note that this does not mean all longs in this range can be encoded - the way IEEE457 floats are implemented means there are "gaps" in the integers that can be encoded as floats once you exceed the number of bits in the mantissa. However, within the newly extended range (which corresponds to the integer range of -2,147,483,583 to 2,147,483,583) the implementation of both ltrunc() and lround() will now return the correct integer value. Outside that range they will return zero.

4. The documentation regarding the 8 Port Serial plugin was a little contradictory regarding the meaning and use of S8_MAX_PORTS. Setting this determines the maximum number of ports that CAN BE OPENED, not the number of ports that will be opened automatically. A port will only be opened automatically if the port number is less than S8_MAX_PORTS, AND it has pins specified in the range 0 .. 63 in the platform configuration file. If the port has pins specified outside this range (e.g. as -1) then the port will not be opened automatically but can be opened manually using the s8_openport() function. A port number greater or equal to S8_MAX_PORTS will never be opened. If you want different baud rates or large buffer sizes, it is recommended to open the ports manually.

5. The accuracy of the various _wait functions (i.e. **_waitsec()**, **_waitms()**, **_waitus()**) has been improved significantly. Also, the calculations of how long each unit of time is in clock ticks are now only done on the first call, not on every call as in previous releases. They are also recalculated on any call where a change in clock frequency is detected, This speeds up the calls, but it means that the very first call in such cases will be very slightly longer than subsequent calls - if this is a problem, add a call like:

   ```
   _waitus(0);
   ```

   to force a re-calculation - subsequent waits will then be more consistent. Note that any call calculates the times for all the _wait functions, not just the one actually called. Also note that for _waitus() there is a minimum number of microseconds that can be waited for accurately - the actual value depends on the memory model used, but for Propeller 2 in NATIVE mode (the default mode) it is about 5 10us, and on the Propeller 1 in TINY mode (the default mode) it is about 30us. If shorter accurate wait times are required, consider using an inline PASM assembly language function instead of the C _wait functions.

6. In previous versions, the 8 port serial plugin would discard the entire receive buffer if a character was received and the buffer was full. It now discards the character received instead if there is no space left in the receive buffer.

7. All the Catalina documents have been revised to correct various typos, and also make the formats more consistent between documents. Also, some have been renamed to make the file names more consistent with the titles and the file contents.

8. The main window of the Windows Installer has been enlarged slightly to ensure all options are displayed (previously, some installation options were only visible if the window was scrolled).

9. The Catalina Command Line menu entry was not being shown in the Windows Start Menu in some versions of Windows - a new batch script called catalina_cmd.bat has been created for this purpose - all it does is open a command window and then execute the use_catalina batch script, which is what the Catalina Command Line start menu entry does.

Release 5.8

New Functionality

1. The Parallax P2 HyperRAM/HyperFlash add on board can now be used either additional RAM & Flash storage or as XMM RAM. It is supported on the P2_EDGE and P2_EVAL boards. There is a new library (**libhyper**) that supports this board, so to use it simply include the **-lhyper** option on the Catalina command line. The hyper library is functionally similar to the existing psram library, but it has additional functions to support the Hyper Flash memory on this board.

   It is not normally necessary to specify the type of XMM RAM when building programs for the P2_EDGE or P2_EVAL. Just specify **-C SMALL** or **-C LARGE** on the command line. On the P2_EVAL Catalina will assume you have a Hyper RAM add-on board, and on the P2_EDGE Catalina will assume you want to use the on-board PSRAM.

   For example - to use Hyper RAM on the P2_EVAL:

   ```
   catalina -p2 -lci -C P2_EVAL -C LARGE hello_world.c
   ```

   Or, to use PSRAM on the P2_EDGE:

   ```
   catalina -p2 -lci -C P2_EDGE -C LARGE hello_world.c
   ```

   However, you can override this by specifying **-C HYPER** or **-C PSRAM** on the command line. For instance, to use the Hyper RAM add-on board on a P2_EDGE, you might say:

   ```
   catalina -p2 -lci -C P2_EDGE -C SMALL -C HYPER hello_world.c
   ```

   Note that you have to use the build_utilities script to build the XMM utilities appropriately. The build_utilities script has been updated to know how to build the XMM utilities required to use the Hyper RAM as XMM RAM. Just specify **HYPER** as the XMM add-on board when building utilities for either the P2_EDGE or the P2_EVAL.

   A simple demo program called *ex_hyper.c* has been added to the Catalina demos\examples folder to illustrate the use of the HyperRAM library.

   A more extensive HyperRAM & HyperFlash test program has been added in a new folder called *demos\p2_ram*, which also contains utilities that can be used to demonstrate PSRAM as well as Hyper RAM.

   Some versions of the P2_EDGE have PSRAM and can use either PSRAM or the Parallax HyperRAM & HyperFlash add-on board, but it is not currently possible to use both the on-board PSRAM and Hyper RAM in the same programs because of name conflicts between the two drivers.

   On the P2_EVAL, the Hyper RAM module uses base pin 32. On the P2_EDGE, it uses base pin 16. You can change this in the files *P2_EVAL.inc* and *P2_EDGE.inc* in the target_p2 directory.
2. Payload has revised file extension processing:
   - Use the bare filename if an extension is specified, but if not, then for the first download try **.binary**, **.eeprom**, **.bin**, **.bix**, **.biy** in that order, and then the bare filename if none of these are found.
   - For the second and subsequent downloads, use the bare filename if an extension is specified, but if not then for a Propeller 1 try **.binary** and **.eeprom** then the bare filename. For a Propeller 2 try **.bin**, **.bix** and **.biy**

then the bare filename.

- Also, unless it has been overridden, set the default baud rate depending on whether the first file has a Propeller 1 or Propeller 2 file extension.
- If a version override is specified (i.e. via command line option -o) then only try the extensions appropriate for the version (**.binary** or **.eeprom** for a Propeller 1, or **.bin**, **.bix**, .**biy** for a Propeller 2) for the first file as well as the second or subsequent files.
- The local directory is always searched for all extensions first, then *%LCCDIR%\bin* (Windows) or *$LCCDIR/bin* (Linux) if no matching file is found in the local directory. This is mainly intended to allow payload to load the various loader utilities, such as XMM, SRAM, EEPROM, FLASH etc.

3. Payload has revised override version processing. If an override version is specified (using the **-o** option on the command line), payload will only detect Propeller chips with the specified version, and will not detect other versions even if they are connected. This allows Propeller 2 **.bin** file to co-exist in the same directory as a Propeller 1 **.binary** files provided the correct override version is specified in the payload command. It also means you no longer have to unplug Propellers if you routinely develop for both Propeller 1 and Propeller 2 chips.
4. **Comms** now correctly decodes the comm port when it is specified on the command line as **/com=N**. Previously, this only worked when N >= 10. It was (and still is) possible to specify it as **/com=comN** on the command line.
5. Blackbox has been modified so that when duplicate source lines have the same address, it associates the address with the LAST such line, rather than the FIRST such line, because this is the line more likely to actually have any code associated with that address.
6. Support has been added in the Propeller 1 target packages for the Parallax Activity board. The Catalina symbol is **ACTIVITY**. This means you can now specify **-C ACTIVITY** on the Catalina command line (or just add **ACTIVITY** to the CATALINA_DEFINE environment variable) and Catalina will use the pin definitions etc in the Activity configuration files (i.e. *Activity_DEF.inc*, *Activity_CFG.inc*, *Activity_HMI.inc*).

   For example:

```
catalina -lc -C ACTIVITY hello_world.c
```

Other Changes

1. The default VGA and USB pins on the P2_EVAL have changed from 32 & 40 to 0 & 8 – this was done to match the P2_EDGE, and also to allow the default base pin used for the P2 HyperFlash & HyperRAM module on the P2_EVAL to be the same as in RogLoh's original Spin drivers – to encourage users of those drivers to try Catalina! :) You can change this in the file *P2_EVAL.inc* in the *target_p2* directory.
2. The *ex_psram.c* demo program in the demos\examples folder has been revised to demonstrate that a program that uses PSRAM for storage can itself be compiled as an XMM program.
3. The **build_utilities** scripts have been extensively re-written to make them more maintainable (e.g. the Linux and Windows versions are now much more similar). The new scripts have been tested for all supported Propeller 2

boards and all the working Propeller 1 boards I still have, but just in case I made a mistake and they do not work correctly, the previous scripts have been left in the utilities folder and called build_utilities_5.7.

Release 5.7

New Functionality

1. Catalina's stand-alone vt100 emulator (**comms.exe**) has an improved ymodem implementation and dialog box:
   - A ymodem transfer can now be aborted in the middle of a send or receive operation.
   - The dialog box now responds to Windows messages. This stops Windows complaining that the application is not responding during a long ymodem transfer.
   - The Abort button now attempts to abort both the local and the remote ymodem applications if a transfer is in progress. Previously, it could only abort the remote ymodem application and only while the local ymodem application was not executing.
   - The Start and Done buttons now check and give an error message if a ymodem transfer is in progress. Previously, they simply performed their respective actions, which could lead to a comms program lock up.
   - Opening the dialog box no longer crashes comms if no port has been opened.
2. Payload's interfile delay has been set to 500ms on Linux. This makes the downloading of XMM programs more reliable on Linux (it was already 500ms on Windows).
3. The **blackbox** source level debugger can now debug XMM programs on the Propeller 2.
4. Some **blackbox** information messages have been changed to make them more accurate.
5. Command line parameters (**argc** &**argv**) were not being correctly initialized when programs were run under the **blackbox** debugger. However, note that it is not possible to pass command-line parameters when using the debugger so they should always be initialized to null values.
6. On all Propeller 2 platforms, the default pins to use for debugging are now pins 62 & 63, which means that if you need to debug a program that uses a serial HMI on those pins, you will need to have a Prop Plug connected to some other pins, and then modify the pins specified in the platform configuration file, which by default are as follows:
   ```
   #define _BLACKCAT_RX_PIN  63
   #define _BLACKCAT_TX_PIN  62
   ```
7. The **blackcat** debugger is no longer supported, and has not been updated to debug Propeller 2 XMM programs. This debugger still works on Propeller 1 programs and non-XMM Propeller 2 programs, but it is now deprecated, and has been removed from the release.
8. The Catalina-specific Geany commands (on the **Build** menu) have been updated slightly to make them less confusing – e.g. '**Build**' has been changed to **Build File** and **Compile** to **Compile File** - these commands

operate on the currently selected file only, and not necessarily the files in the project directory (even thought the currently selected file may have the same name as a project file!). The **Link** command has been changed to **Link All** to match **Compile All**, to indicates that these commands act on all files in the project directory. Other changes ensure that any output generated when a project is selected end up in that project directory.

9. Payload now accepts a baud rate of zero (i.e. **-b0** or **-B0**) to indicate that the default baudrate appropriate for the propeller should be used (which is 115200 on a P1 or 230400 on a P2). This was the default if NO baudrate was specified, but Geany HAS to have a baud rate specified, so specifying it as 0 is now allowed, and is equivalent to not specifying it. All the Geany projects have been updated to use baud rate 0

Other Changes

1. The compiler now reports the correct code size when compiling XMM LARGE programs.

Release 5.6

New Functionality

1. Catalina can now build and execute XMM SMALL and XMM LARGE programs on Propeller 2 platforms with supported XMM RAM. Currently the only supported type of XMM RAM is PSRAM, and the only supported platform is the P2_EDGE, but this will expand in future. To compile programs to use XMM RAM, simply add **-C SMALL** or **-C LARGE** to a normal Catalina compilation command.

   For example:

   ```
   cd demos
   catalina -p2 -lci -C P2_EDGE -C SMALL hello_world.c
   ```

   or

   ```
   cd demos
   catalina -p2 -lci -C P2_EDGE -C LARGE hello_world.c
   ```

   Note that a 64kb XMM loader is always included in the resulting binaries, so the file sizes will always be at least 64kb.

2. The payload serial program loader can now load XMM programs to the Propeller 2. Loading XMM programs requires a special XMM serial loader to be built, which can be built using the **build_utilities** script. Currently, the only supported type of XMM RAM is PSRAM, and the only supported P2 platform is the P2_EDGE. The script will build *SRAM.bin* (and *XMM.bin*, which is simply a copy of *SRAM.bin*). These load utilities can then be used with payload by specifying them as the first program to be loaded by the payload command, and the XMM program itself as the second program to be loaded.

   For example:

   ```
   cd demos
   catalina -p2 -lc -C P2_EDGE -C LARGE hello_world.c
   build_utilities                <-- follow the prompts, then ...
   payload xmm hello_world.bin -i
   ```

3. The Catalyst SD loader can now load XMM programs from SD Card on the

Propeller 2. Just build Catalyst as normal, specifying a suppored P2 platform to the 'build_all' script. Currently, the only type of XMM RAM supported is PSRAM, and the only supported P2 platform is the P2_EDGE.

For example:

```
cd demos\catalyst
build_all P2_EDGE SIMPLE VT100 CR_ON_LF
```

Note that now that Catalyst and its applications can be built as XMM SMALL or LARGE platforms on the Propeller 2, it is no longer possible to specify a memory model on the command line when building for Propeller 2 platforms. This is because some parts of Catalyst MUST be build using a non-XMM memory model. However, it is possible to rebuild the non-core component of Catalyst as XMM SMALL or XMM LARGE programs. See the *README.TXT* file in the *demos\catalyst* folder for more details.

4. New functions have been added to start a SMALL or LARGE XMM kernel:

```
_cogstart_XMM_SMALL();
_cogstart_XMM_SMALL_cog(g);
_cogstart_XMM_LARGE();
_cogstart_XMM_LARGE_cog();

_cogstart_XMM_SMALL_2();
_cogstart_XMM_SMALL_cog_2();
_cogstart_XMM_LARGE_2();
_cogstart_XMM_LARGE_cog_2();
```

These are supported on the Propeller 2 only. The **_2** variants accept two arguments rather than just one, to allow the passing of **argc** and **argv** to the program to be started.

Refer to *catalina_cog.h* for details of parameters etc.

5. The Cache and Cogstore cogs are now explicitly registered and can now therefore be listed by a C program interrogating the register (e.g. by the demo program *ex_registry.c*)
6. The p2asm PASM assembler now allows conditions to be specified as **wc,wz** or **wz,wc** which both mean the same as **wcz**, It also does more rigorous checking about which instructions should allow such conditions.
7. **OPTIMISE** can now be used as a synonym for **OPTIMIZE**. Both enable the Catalina Optimizer when used in **CATALINA_DEFINE** environment variable or as a parameter to one of the build_all scripts. For Catalina command-line use you can use either **-C OPTIMISE** or **-C OPTIMIZE** or the command-line argument **-O5**.
8. **COLOUR_X** can now be used as a synonym for **COLOR_X**, where **X** = 1, 4, 8 or 24. Both specify the number of colour bits to be used in conjunction with the VGA HMI options. For example, on the Catalina command-line you could use either **-C COLOUR_24** or **-C COLOR_24**.
9. The **build_catalyst** script now looks for Catalyst in the current directory first, then the users home directory (for folder *demos\catalyst*) before finally resorting to building Catalyst in the Catalina installation directory.
10. The **build_utilities** script now looks for utilities to build in the current directory first, then the users home directory (for folder *utilities*) before finally resorting to building the utilities in the Catalina installation directory.
11. There are now more pre-built demos of Catalyst. In the main Catalina

directory you will find:

P2_DEMO.ZIP	compiled to suit either a **P2_EVAL** or a **P2_EDGE** without PSRAM, using a serial HMI on pins 62 & 63 at 230400 baud. Built to use a VT100 emulator, or **payload** with the **-i** and **-q1** options.
P2_EDGE.ZIP	compiled to suit a **P2_EDGE** with 32 Mb of PSRAM, using a serial HMI on pins 62 & 63 at 230400 baud. Built to use a VT100 emulator, or **payload** with the **-i** and -**q1** options.
P2_EVAL_VGA.ZIP	compiled to suit a **P2_EVAL** using a VGA HMI with VGA base pin 32 and USB base pin 40.
P2_EDGE_VGA.ZIP	compiled to suit a **P2_EDGE** with 32Mb of PSRAM using a VGA HMI with VGA base pin 0 and USB base pin 8.

In all cases, unzip the ZIP file to an SD Card, and set the pins on the P2 board to boot from the SD Card.

Other Changes

1. Fixed a bug in the C clock() function, which could return incorrect values. Affected both the Propeller 1 and Propeller 2.
2. The **vtXXX** scripts (e.g. **vt100** or **vt100.bat**) now start the emulator with the wrap option specified, to match payload's built in terminal emulator.
3. The Pascal P5 compiler and interpreter are now explicitly started with **-C CLOCK** to ensure that the clock plugn is started.
4. On the Propeller 2, the Super Star Trek demo now specifies the maths library as **-lmc** instead of **-lma**. No functional difference, but this makes the binary smaller when compiled in XMM SMALL mode on the Propeller 2.
5. To enable the dynamic XMM kernels to be built, the **spinc** program has been updated to allow it to extract data from files that identify themselves as XMM SMALL or XMM LARGE. However, this is not full XMM support – it is only sufficient support to allow the code segment to be extracted from such files when built as part of the Catalina library, and these are not true XMM binary files.

Release 5.5.2

New Functionality

1. The Catalyst vi editor (xvi) demo program has been updated to version 2.51, which is the latest version and contains some new functionality and some bug fixes. See the documents in *demos\catalyst\xvi-2.51\doc* for details.
2. The Catalyst vi editor now processes the keypad keys **PgUp**, **PgDown**, **Home**, **End**, **Ins** & **Del** appropriately.
3. The Catalyst vi editor can now be compiled to use colour when the VT100 HMI option is used, since colour is supported by the new external terminal emulator (**comms.exe**). Just define the Catalina symbol **USE_COLOR** in addition to **VT100**, either on the Catalina command line or by adding it as a parameter to the build_all command. For example:

```
cd demos/catalyst/xvi-2.51
```

```
build_all P2_EDGE CR_ON_LF VT100 USE_COLOR
```

Note: Only foreground colours can be set within vi itself – the current background colour (typically the background colour in use before vi was started) will be used. The default colours are that text will be grey, the status line will be green (normal files) or red (read-only files). Try the vi command **:help** to see an example.

You can manually set the colours (for the text, status line, or read-only status line) in vi as follows:

:set colour=n

:set statuscolour=n

:set roscolour=n

where n is:

0 : black
1 : red
2 : green
3 : yellow
4 : blue
5 : magenta
6 : cyan
7 : grey
9 : default foreground colour
10 : bold and bright black (dark grey)
11 : bold and bright red
12 : bold and bright green
13 : bold and bright yellow
14 : bold and bright blue
15 : bold and bright magenta
16 : bold and bright cyan
17 : bold and bright grey (white)

Note that when vi is compiled to use colour, the default vi colours will override the current foreground colour on startup. To get it back again, use the vi command **:set colour=9**

Any value other than those specified will set the colour to the default foreground colour (typically, the foreground colour before vi was started). If you are using **comms.exe** then you can also select the default colours using the Format menu (remember to select the **Format->Set All To Colours** menu item after choosing your new colours, or the changes will apply only to new characters, not existing ones).

Note that you can also have other effects, such as inverse video - (i.e. black text on white bg). To do this, start vi and then set the foreground colour to black (i.e. **:set colour=0**) - note that any text on the screen may temporarily disappear. Then use the **Format->Bg Color** menu item to select a white background, then use the **Format->Set All to Colour** menu item to refresh the colours of the characters already on display. You can choose to have any background colour you want.

Other Changes

1. A change to the clock and SD Card plugin code (made in release 5.4) means that the Catalyst Super Star Trek demo now needs to be explicitly built with the **-C CLOC**K option. The Makefile has been modified accordingly.
2. When the external terminal emulator's YModem dialog box is open, it is now always on top. Otherwise, it could get hidden by other windows and it was not obvious why the main comms window would not respond.
3. The external terminal emulator's YModem dialog box **Start** button is no longer styled as if it was a default button, since it was not. There is no default button for this dialog box.
4. The new external terminal emulator now locks the screen and view. This makes it easier to set the initial screen size (can now be done just using the **/ScreenSiz**e option, instead of also requiring the **/ViewSize** option). This is more appropriate when emulating a physical terminal, where the screen size would normally be the same as the view size. They can be unlocked again (if required) via the **Options->Advanced** menu.
5. The scripts that call the external terminal emulator (vt100.bat etc) now add **/Wrap** to enable wrap mode on startup. This option can also be changed via the **Options→Advanced** menu. Setting it on by default makes the external terminal emulator behave more like the internal terminal emulator.
6. The payload internal terminal emulator now limits the number of restart retries on ymodem_receive, to avoid locking up permanently in case the other end has not been started.
7. The file *P2_EVAL.ZIP* has been renamed *P2_DEMO.ZIP*, and the *README_P2.TXT* file has been updated to indicate that this demo version of Catalyst should work on the P2 EVAL, P2 EDGE or any other P2 board with a similar SD Card configuration.

Release 5.5.1

New Functionality

1. The Windows version of Catalina now includes a new serial comms program with a full-function terminal emulator, which can be used as a replacement for payload's very simple interactive mode by specifying **-I terminal** instead of just **-i** on the payload command line – see below for more details).

   The binary for the program (**comms.exe**) is in Catalina's *bin* folder, and the source is in *source**comms*, but note that the source is not built by default if Catalina is rebuilt, since it requires GNAT (the GNU Ada Translator) and the GNAT Studio development environment to be installed. See the file BUILD.TXT in *source**comms* for details on how to rebuild the **comms** program from source.
2. Payload now allows an external terminal emulator program to be executed in place of the simple internal one. The **-i** option still enables the internal emulator, but a new **-I** option (i.e. upper case) calls an external terminal emulator program. The **-I** option accepts one parameter which specifies the command to be used. The port name (not the port number) and baud rate will be passed to this program on startup.

   For example, if the payload command executed was:

```
payload program.bin -p11 -b230400 -I vt100
```

Then after loading the **program.bin** file, the command executed (on Windows) would be:

```
vt100 COM11 230400
```

On Linux, the command would be something like:

```
vt100 /dev/ttyUSB0 230400
```

A suitably named script (i.e. on Windows a batch file called **vt100.bat**, or on Linux a shell script called **vt100**) can be used to specify any other required parameters.

The following Windows scripts are provided (each one is a batch file):

```
pc              <-- start comms, specifying PC emulation
vt52            <-- start comms, specifying VT52 emulation
vt100           <-- start comms, specifying VT100 emulation
vt101           <-- start comms, specifying VT101 emulation
vt102           <-- start comms, specifying VT102 emulation
vt220           <-- start comms, specifying VT220 emulation
vt320           <-- start comms, specifying VT320 emulation
vt420           <-- start comms, specifying VT420 emulation
vt100_putty     <-- start PuTTY, specifying VT100 emulation
```

The following Linux scripts are provided (each one is a shell script):

```
ansi            <-- start minicom, specifying ANSI emulation
vt100           <-- start minicom, specifying VT100 emulation
vt102           <-- start minicom, specifying VT102 emulation
```

Note that these scripts can be used independently of payload. They all accept two parameters – the com port to use, and the baud rate. For example:

```
vt100 COM11 230400
vt102 /dev/ttyUSB1 115200
```

Note that the **comms** program executable is provided as part of the Windows version of Catalina, but **PuTTY** and **minicom** executable will have to be installed separately. Ensure the appropriate executable is in the PATH.

Other Changes

1. Payload now interprets either **ESC [? 6 n** or **ESC [6 n** as a DEC DSR request and responds with a DEC CUP. Strictly speaking, **ESC [? 6 n** is not a correct DEC DSR command, and would not be recognized by other terminal emulators, so the Catalyst xvi program now uses the more correct **ESC [6 n**

Release 5.5

New Functionality

1. A version of the YModem serial file transfer protocol has been added to Catalyst. When Catalyst is built, there will be two more Propeller programs added to the *demos\catalyst\bin* directory:

   send.bin – send a file from the Propeller to the host.

   receive.bin – receive a file on the Propeller sent from the host.

   Also, in the *demos\catalyst\fymode*m directory, there will be two corresponding host binaries (on Windows these will be .exe files):

   send – send a file from the host to the Propeller

receive – receive a file on the host sent from the Propeller

See the *README.TXT* file in *demos\catalyst\fymodem* for more details.

2. A version of the YModem serial protocol has been integrated into payload's interactive mode. This can be used for file transfers even if the Propeller does not use a serial HMI option.
3. Payload now supports configuring the line termination mode from within interactive mode as well as on the command line (i.e. using the **-q** command line option).
4. Payload now has menus. Pressing the "attention" key (**CTRL-A** by default) in payload's interactive mode now brings up a menu from which you can select either Terminal Configuration, YModem Transfer, or to Exit from payload (as an alternative to pressing **CTRL-D** twice).
5. The *ex_leds_1.c* and *ex_leds_2.c* example programs in demos\examples now support the **P2_EDGE** board.
6. Payload now responds correctly to the DEC VT100 Cursor Position Requests (CUP) and responds to a Device Status Report (DSR) specifying parameter 6 with a Cursor Position Report (CPR). Taken together, this allows programs expecting a VT100 compatible terminal to determine the actual terminal window size, rather than simply assuming it will always be 80×24.
7. The Catalyst vi program now uses the DEC VT100 CUP and DSR requests to elicit a CPR, which allows it to determine the actual window size. This means that vi is no longer limited to 80×24 windows when a serial HMI option is used. In particular, when payload is used, the actual window size will now be reported correctly to vi on startup, and vi will use it rather than defaulting to 80×24 whenever a serial HMI is used. The default VT100 window size (80×24) will still be used if VT100 is specified in the build and the terminal emulator cannot respond to CUP and DSR requests.
8. For example, to use vi in a terminal window of 50 lines and 120 columns, and assuming you are using a serial HMI and specified the VT100 option when vi was built, then you might start payload as follows (assuming N is the comms port connected to your Propeller):

```
payload -i -g120,50 -b230400 -pN
```

Then within payload just start vi as normal and it will use the correct window size – e.g:

```
vi catalyst.txt
```

Other Changes

1. The files *s2_serial.h*, *s4_serial.*h and *s8_serial.*h were not correctly defining the symbols **s2_getc**, **s4_getc** and **s8_getc** respectively. These names were only intended to be used to emulate the original Spin names – the actual underlying C functions (i.e. **s2_rxcheck**, **s4_rxcheck** and **s8_rxcheck** respectively) were all fine.
2. The Catalina Optimizer may have reported undefined symbols when optimizing programs that used the serial2 or serial8 libraries.
3. The header files for the *tty*, *tty256*, *serial2*, *serial4* and *serial8* libraries now check and issue a message (using #error) if the libraries they represent are not supported on the Propeller type for which the program is being compiled:
   - The *serial2* and *serial8* libraries are only supported on the Propeller 2.

- The *tty*, *tty256* and *serial4* libraries are only supported on the Propeller 1 (but note this has nothing to do with the **TTY** HMI option, which applies to both the Propeller 1 and Propeller 2 – the b HMI option does not use these libraries).

4. The rs232 interface module (*rs232.h*, *rs232.c*) used by payload and fymodem included code intended to prevent DTR being toggled when opening a serial port. This code no longer works correctly on Debian Linux and has been disabled. If it is required on other platforms, it can be re-enabled by setting **rs232_DTR_FIX** to 1 in the file *rs232.h*. Note that copies of this file exist in both the folder *source/catalina* as well as the folder *demos/catalyst/fymodem* folder, and will need to be modified in both places.
5. The help text for payload's **-q** command line option had the values for enabling **CR_TO_LF** and **LF_TO_CR** translation the wrong way around.

Release 5.4.1

New Functionality

6. **p2asm** now supports the "IF_NN" and "IF_NOT_NN" (where N=0,1 or X) PASM instruction prefixes, as well as "IF_SAME" & "IF_DIFF". For example, the following two statements will generate the same PASM instruction:

```
if_c_or_z  mov r0, #1
if_not_00  mov r0, #1
```

Other Changes

1. Programs compiled in SMALL mode and loaded using Catalyst were not setting up the argc and argv parameters to the C main function correctly. Affected the Propeller 1 only.
2. The *arg.c* argument diagnostic program in the utilities folder was not working correctly on the Propeller 1.
3. It was not clear in the Catalyst documentation that when Catalyst is used to load XMM programs, then both Catalyst and the XMM programs must use the same caching and flash options. For example, if Catalyst is compiled using FLASH CACHED_1K then so must all XMM programs (but note that they can be SMALL or LARGE programs). This only applies to XMM programs, and not to LMM, CMM, SMM, Spin or Lua programs. Applies to the Propeller 1 only.
4. The Catalina Optimizer was failing to perform some optimizations when the optimization level was 3 or greater, so that -O3 could result in a larger code size than -O2. Affected all memory models except COMPACT on both the Propeller 1 and the Propeller 2.

Release 5.4

New Functionality

1. Roger Loh's 16 Bit PSRAM driver has been added as a Catalina plugin. It is supported on the Propeller 2 only. It is enabled by linking with the new psram library (i.e. adding **-lpsram** to the Catalina command). An example of its use has been added in *demos\examples\ex_psram.c*. The configuration parameters for the driver must be specified in the platform files in the

target_p2 directory, such as P2_EDGE.inc.

You would compile the PSRAM example program with a command like:

```
catalina -p2 -lc -lpsram -C P2_EDGE ex_psram.c
```

The only tested platform is the P2 EDGE, and like the driver itself, Catalina's support for it should be considered a beta version until further notice.

2. As a demonstration of the use of the PSRAM plugin, Lua now has the option to store code in PSRAM on those platforms that support it, such as the P2 EDGE. This allows larger Lua programs to be executed at the cost of a slight speed reduction. Supported on the Propeller 2 only.

   To enable the use of PSRAM in Lua, specify **ENABLE_PSRAM** to the Catalyst or Lua build_all scripts. For example:

   ```
   build_all P2_EDGE SIMPLE VT100 ENABLE_PSRAM
   ```

   Note that PSRAM is supported only by the Lua execution engine (**luax**) which executes compiled Lua programs, and not for the interactive version (**lua**) that executes text programs or the Multiprocessing version (**mlua** or **mluax**). So if **ENABLE_PSRAM** is specified, only **luax** will be built. This means you may need to build Lua twice – once to build the lua programs that do not use PSRAM, and then again to build **luax** (only) to use PSRAM.

   For example, to compile Lua in directory *demos\catalyst\lua-5.4.4* and put the executables in *demos\catalyst\bin* and call the PSRAM version **luaxp** rather than overwrite the standard **luax** binary, you might use commands such as:

   ```
   cd demos\catalyst\lua-5.4.4
   build_all P2_EDGE SIMPLE VT100
   copy src\*.bin ..\bin\
   build_all P2_EDGE SIMPLE VT100 ENABLE_PSRAM
   copy src\luax.bin ..\bin\luaxp.bin
   ```

   Note that specifying **ENABLE_PSRAM** is applicable only when using the Catalyst and Lua build_all scripts and Makefiles – it is not a general Catalina symbol that can be used on the Catalina command-line to enable PSRAM in other cases (which is done via the usual mechanism of linking the program with the psram library – i.e. adding **-lpsram** to the catalina command).

   Finally, note that for small Lua programs, the Hub RAM usage of the PSRAM version may not be much smaller than that of the non-PSRAM version – it may even be larger. This is not only because of the additional PSRAM support code required, it is also because the PSRAM version allocates a fixed amount of Hub RAM on startup to use as a PSRAM cache, and for small Lua programs the cache may be larger than the program being loaded. However, the amount of Hub RAM used for Lua code will never increase beyond the cache size no matter how big the program code gets.

3. Catalyst on the Propeller 1 can now use the **HIRES_VGA** option. However, Hub RAM limitations mean that Catalyst itself needs to be built as an **EEPROM** program, and some of the Catalyst utilities may only work if they are built as **LARGE** programs.

   This means that while Catalyst itself will work in **HIRES_VGA** mode on all platforms, some utilities (such as **cp** & **mv**) may only work in **HIRES_VGA** mode on platforms with XMM RAM. Alternatively, you might build Catalyst itself in **HIRES_VGA** mode, but the utilities in **LORES_VGA** mode.

To facilitate this, two new options have been added that can be used with the Catalyst build_all scripts:

EEPROM_CATALYST	specifies that the Catalyst binary should be built as an EEPROM program.
LARGE_UTILITIES	specifies that the Catalyst utilities should be built as LARGE programs.

Whether you need to specify one or both of these options can depend on the other options used. For instance, if you need to use the cache to access XMM RAM, then you will generally need to use both of these options to build Catalyst in **HIRES_VGA** mode. For example, here is how you might build Catalyst to use **HIRES_VGA** on the C3:

```
build_all C3 FLASH CACHED_1K HIRES_VGA EEPROM_CATALYST LARGE_UTILITIES
```

When you build Catalyst to use **LORES_VGA** or **HIRES_TV** HMI option, you may also find that *catalyst.binary* exceeds the size of Hub RAM and in that case you can specify the **EEPROM_CATALYST** option, but you may not need the **LARGE_UTILITIES** option – e.g:

```
build_all C3 FLASH CACHED_1K HIRES_TV EEPROM_CATALYST
```

Note that to program Catalyst into EEPROM when the **CATALYST_EEPROM** option is used, you will need to run the **build_utilities** script to build the EEPROM loader, and then use that with payload. For example, to build and load Catalyst to use **HIRES_VGA** on the C3, you might use commands like:

```
cd demos\catalyst
build_all C3 FLASH CACHED_1K HIRES_VGA EEPROM_CATALYST LARGE_UTILITIES
build_utilities
payload -o1 EEPROM ..\bin\catalyst.bin
```

Note that these new options are applicable only when using the Catalyst **build_all** scripts and Makefiles – they are not Catalina symbols that can be used in other circumstances (i.e. specifying **-C EEPROM_CATALYST** when compiling *catalyst.c* manually will not have any effect. It is the Makefile that intercepts this symbol and instead uses **-C EEPROM**, but only when building *catalyst.binary*).

4. The Catalyst ONCE capability (i.e. to execute a command once on reboot) has been extended to execute MORE than one command. If enabled when building Catalyst (by setting both **ENABLE_ONCE** and **ENABLE_MORE** to 1) then the file (*EXECONCE.TXT* by default) can contain more than one command. The commands in the file will be executed in sequence on successive reboots. The Lua **execute** function can be used to easily add multiple commands to the file, one per line. For example, if the MORE capability is enabled, the Lua statement:

   ```
   propeller.execute("vi abc\n vi def")
   ```

   will cause the propeller to first reboot and execute the command **vi abc**, and then when vi exits, the propeller will reboot and execute the command **vi def**.

   This also allows a very basic scripting capability to be implemented. For example, if you have a file called *commands.txt* which contains all the commands you want executed, then executing the command:

   ```
   cp command.txt execonce.txt
   ```

   at the Catalyst prompt, or executing the Lua statement:

```
propeller.execute("cp command.txt execonce.txt")
```

will cause all the commands in the file to be executed in sequence.

Note that this capability is enabled by default on the Propeller 2, and on the Propeller 1 unless the **HIRES_VGA** HMI option is used, because there is not enough Hub RAM available. If you want to enable it, you may need to disable something else, such as the capability to allow Lua commands to be executed directly from the command line (note that you can still execute them by specifying them as parameters to Lua – e.g. by entering a command like **lua list.lua** instead of just **list**). This can be disabled by editing *demos\catalyst\core\catalyst.h* and rebuilding Catalyst.

5. The internal **cat**, **dir** and **help** commands have been removed from Catalyst. The help and cat commands are now always external. The **dir** command has simply been dropped since the **ls** command is far more capable. However, if you prefer typing **dir** to **ls**, then just make a copy of **ls.bin** called **dir.bin** – i.e:
   ```
   cp ls.bin dir.bin
   ```
6. Added a new Lua demo called *wild.*lua that can be used to add wildcard capability to a Catalyst command that accepts multiple file names. For example, if the command specified in *wild.lua* is "vi" (which it is by default) then
   ```
   luac -o xvi.lux wild.lua
   ```
   will create a command **xvi** which can then be used on the command line to invoke vi on multiple files – e.g:
   ```
   xvi ex*.lua
   ```
   will execute vi on all the Lua example files.
7. The length of a Catalyst command line has been increased to 300 characters on the Propeller 1, and 1200 characters on the Propeller 2. Hub RAM is limited on the Propeller 1, but 300 is enough to accommodate 24 MSDOS 8.3 filenames, which could be generated when using wildcards, and 1200 is the maximum that CogStore can store. This maximises the potential usefulness of the wildcard functionality added in the last few releases.
8. A new **getrand()** function has been added to the C library. It is defined in *propeller.*h and is implemented on both the Propeller 1 and the Propeller 2 (but differently – see below). A program that demonstrates the use of the function has been added in *demos\examples\ex_random.c*

   On the Propeller 1, the first time this function is called it calls **srand()** with the current **CNT** value and is therefore best called after some user input or other random source of delay. It then returns the result of 3 combined calls to rand() to make up 32 random bits (rand itself only returns 15 bits).

   On the Propeller 2, the first time this function is called it calls **srand()** with the result of the **GETRND** opcode, and also returns that value. Thereafter it just returns the result of the **GETRND** opcode.

   This means you can either use just this function, or use this function once to generate a seed for **srand()** and thereafter use rand(), which is what most traditional C programs would typically do.

   Note that rand() only returns a value between 0 and **RAND_MAX** (inclusive) (i.e. 0 .. 32727 on the Propeller) whereas getrand() returns 32 bits. To simulate **rand()** using **getrand()**, use an expression like:
   ```
   (getrand() % (RAND_MAX + 1))
   ```

9. On the P2_EDGE, the base pin for VGA has been changed to 0 (was 32), and the base pin for USB has been changed to 8 (was 40). This makes it possible to use the P2-ES VGA and USB accessory boards with Catalina on the P2_EDGE. The base pins can be edited if required in the file *P2_EDGE.inc* in the *target_p2* folder.

Other Changes

1. Fixed a bug in Catalina's code generator that meant statements like "x = x op y", where x was a complex lvalue (e.g. an element of an array) and op was + or – could cause the compilation to fail. Affected both the Propeller 1 and Propeller 2.
2. Fixed a bug in p2asm, which was not correctly processing a count of zero in **BYTE**, **WORD** or **LONG** statements – e.g:
```
pad LONG 0[COUNT]  ' should not generate data if COUNT is zero
```
   Affected the Propeller 2 only.
3. Fixed a bug in p2asm, which did not understand the "!" operator (which should be
```
MOV    r0,##!$21524110   ' should put $DEADBEEF into r0
```
   Affected the Propeller 2 only.
4. Multi-Processing Lua example 12 (*ex12.lua*) has been reduced from using 5 factories (i.e. cogs) to 4. 5 cogs were not always available, depending on which HMI plugins that were loaded. Affected the Propeller 2 only.
5. Lua example *list.lua* was using the wrong value for the DOSFS "archived" file attribute. Affected both the Propeller 1 and Propeller 2.
6. Catalyst was supposed to be able to accept up to 24 command line arguments but in fact CogStore would only correctly process the first 12. Affected both the Propeller 1 and the Propeller 2.
7. Fixed an issue in some Makefiles intepreting the **-p2** flag when it was specified via the **CATALINA_OPTIONS** environment variable (which is mostly used by the Geany IDE to pass options to the Makefile). Affected the Propeller 2 only.
8. The Propeller 2 *Catalina_platforms.inc* configuration file was getting a bit unwieldy, so the platform constants have now been split into a separate file for each platform. The *Catalina_platforms.inc* file still exists but it now just includes one of the following include files to define the platform constants:

   P2_EVAL.inc

   P2_EDGE.inc

   P2D2.inc

   P2_CUSTOM.inc

   These are all in the target_p2 directory. The default if no platform is specified is to use *P2_EVAL.inc*, but this can be changed by editing *Catalina_platforms.inc*.
9. Lua 5.1.5, which was deprecated in previous releases, has been removed altogether from this release.
10. Some of the Catalyst demo programs were either not specifying **NO_MOUSE**, or were using **-lmb** when they could have used -lma. In most

cases this was fine, but it meant some were using additional cogs unnecessarily, which prevented them running in **HIRES_VGA** mode (which takes one more cog than the other HMI options on the Propeller 1). Affected the Propeller 1 only.

11. On the Propeller 1, enabling the **CLOCK** no longer requires an extra cog when the SD card plugin is also loaded. This was the case in early Catalina releases, but it has not been the case since the new SD card plugin was adopted, due to lack of code space in the newer plugin. This has been fixed by adding a new (smaller) clock service and shifting most of the time calculations to the C library instead. Support for the old SD card plugin (which could be enabled by defining the Catalina symbol **OLD_SD**) has therefore been dropped, because that was the only reason it was still included. The new clock service has been added on both the Propeller 1 and the Propeller 2 for consistency, even though it was not necessary on the Propeller 2 because its SD plugin does have sufficient space to implement the old clock services. You can force the inclusion of a separate clock plugin even when the SD Card plugin is loaded by specifying the Catalina symbol **SEPARATE_CLOCK**. This would be necessary if you have a custom **CLOCK** plugin (e.g. one that uses a hardware clock).
12. The Catalina optimizer was still assuming it could use the Homespun spin compiler, which is no longer included with Catalina. It has been modified to use Spinnaker/OpenSpin in all cases.
13. The Lua demo programs and examples now pause before terminating, so that if they are used with a non-serial HMI option (such as VGA) the output is not lost before it can be read (Catalyst clears the screen when the program terminates).
14. The Propeller Memory Card (PMC) had a timing issue which may have caused the card initialization to fail. Added NOPs to extend the clock pulses. Affected the Propeller 1 only.
15. If you chose not to copy the utilities to the Catalina bin directory, the build_utilities script said it would build the utilities in the "current" folder, but what it meant was the "utilities" folder, assuming that was where it was executed. In fact it can be executed in any directory, but will always either leave the utilities in the utilities folder or copy them to the bin directory. The text has been updated. Affected the Propeller 1 only.

Release 5.3.1

New Functionality

1. Catalyst now understands ".lux" as a filename extension, as well as ".lua". Catalyst assumes ".lux" represents a compiled Lua script, whereas ".lua" can represent either a compiled or a non-compiled Lua script. By default, files with a ".lux" extension will be executed with LUAX.BIN and files with a ".lua" extension will be executed with LUA.BIN. If no extension is specified on the command line and files with both extensions exist, ".lux" will be used.
2. There is now a Propeller 2 version of the Catalyst Spin demo that shows how to use Catalyst command-line arguments in Spin2 files. It is in the folder *demos/catalyst/demo* and called *Demo.spin2*.

Other Changes

1. Fixed a race condition in the SD Card plugin, which meant that it could fail when a sequence of SD Card operations was performed at a specific rate. Affected the Propeller 2 only.
2. The demo program *ex_time.c* was not clearing the daylight savings flag when setting the time, which meant that when the time was retrieved, the hour would sometimes be one hour different than what was set. Affected both the Propeller 1 and Propeller 2.
3. Dumbo Basic was not flushing its output buffers, so prompts and input was not appearing in some cases when executing input statements until an end-of-line character was entered

Release 5.3

New Functionality

1. The Catalyst utilities **cp**, **mv**, **rm**, **ls** and **cat** now ignore volume id entries, except that the ls utility will still include them when long or very long listing output is specified.
2. Catalyst now has the ability to execute a once-only command on startup. This is similar to the existing AUTOEXEC.TXT file processing when the AUTODELETE option was also enabled. The AUTOEXEC.TXT and AUTODELETE functionality has been retained, but AUTODELETE is not enabled by default, which means that normally the AUTOEXEC.TXT is executed on every Propeller reboot. The new once-only execution functionality overrides this if the file EXECONCE.TXT exists. If it does then it will be executed and also deleted. This new functionality is used by the Lua propeller module "execute" function, described in point 3 below. It requires that the SD Card be writable.
3. The Lua propeller module now includes new propeller-specific functions.

   Lua analogues of the C DOSFS wildcard/globbing functions:

   ```
   mount()
   scan(function, directory, filename)
   ```

   The filename can include wildcards. See the *Catalyst Reference Manual* for details, and see *list.lua* for an example of using these functions.

   A Lua function to execute any Catalyst command (reboots the Propeller):

   ```
   execute(command, filename)
   ```

   For example:

   ```
   propeller.execute("list *.bas")
   ```

   The first parameter is the command (which may include parameters), and the second (optional) parameter is the file name to write this command to. This defaults to "EXECONCE.TXT", which means the command will only be executed once, but it can be used to write the command to any file. For instance:

   ```
   propeller.execute("lua", "AUTOEXEC.TXT")
   ```

   will cause the Propeller to execute **lua** on each reboot. To disable this from within Lua, just delete the file by executing a Lua command like:

   ```
   os.remove("AUTOEXEC.TXT")
   ```

   Lua analogues of the keyboard HMI functions:

```
k_get, k_wait, k_new, k_ready, k_clear
```

Lua analogues of the screen HMI functions:

```
t_geometry, t_mode, t_setpos, t_getpos, t_scroll,
t_color, t_color_fg, t_color_bg
```

Lua analogues of the mouse HMI functions:

```
m_button, m_abs_x, m_abs_y, m_delta_x, m_delta_y, m_reset,
m_bound_limits, m_bound_scales, m_bound_x, m_bound_y
```

Each of these HMI functions accepts the same parameters and returns the same values as their C counterparts. See the *Catalina Reference* Manual for details.

Note that to save space, the mouse functions will not be included if the Catalina symbol NO_MOUSE is defined when Lua is compiled (which it will be if you use the **build_all** scripts to build Lua – to change this you can edit the file **Makefile.Catalina** to remove the **-C NO_MOUSE** option). You can detect whether mouse functions have been included from within Lua itself by testing if any of them are nil, such as:

```
if (propeller.m_button) then
   -- mouse functions have been included
else
   -- no mouse functions
end
```

4. Catalyst will now try to execute commands as Lua scripts, in addition to just executing binary files. The order of priority for Catalyst command execution is now:
    1. As a built-in command (e.g. **dir**)
    2. As a Lua script, adding a ".lua" extension if none is specified
    3. As a binary file, using the command as the file name
    4. As a binary file, adding the following extensions (in this order if the command does not specify an extension):

        BIN

        BIX (propeller 2 only)

        XMM

        SMM

        LMM

    Note that all command types accept command line arguments, including Lua scripts. Several example Lua scripts are included:

    list.lua – a simple directory listing program (similar to Unix **ls**).

    find.lua – a simple file searching program (similar to Unix **find**).

    freq.lua – a simple word frequency counting program.

    You invoke Lua scripts from the Catalyst command line just like any other command. For example:

```
list                <-- list file details of all files
list *.lua          <-- list file details of Lua files
find *.bas PRINT    <-- find PRINT statements in all files
freq *.txt          <-- count word frequency in all text files
```

    Note that Lua scripts can be compiled to improve load and execution times, but they should still have the extension ".lua".

As a consequence of this, the Lua versions of the Super Star Trek demo programs have changed name, so that executing the command **sst** at the Catalyst prompt will still invoke the C version (**sst.bin**) rather than now executing the Lua version (which used to be called **sst.lua**):

```
sst.lua         --> now called star.lua
sst-tos.lua     --> now called star-tos.lua
```

5. The Propeller 2 Catalina command line argument processing has been modified to match that of the Propeller 1. On startup, all Catalina programs check if CogStore is running. If it is, the program fetches any arguments stored in it. Previously on the Propeller 2, if CogStore was not running then Catalina would set **argc** to zero and **argv[0]** to **NULL** – but some C programs expect argc to always be at least 1, so now if CogStore is not running **argc** will be set to 1 and **argv[0]** will point to a string with the value "null" (since without CogStore the real program name is not available). This functionality was already implemented on the Propeller 1.
6. 6. The wildcard/globbing **doDir()** function in *storage.h* and *storage.c* has been extended to make it more useful – it now calls the file processing function with the file size and the DOSFS file attributes of each file in addition to the file name. One advantage of this is that programs that use **doDir()** do not need to use DOSFS functions to retrieve the file attributes themselves, and therefore only need to use standard C stdio file functions.
7. The **build_catalyst** script now detects whether it is being run in the current directory. If so, it builds catalyst in this directory, otherwise it builds it in the demos\catalyst folder in the Catalina installation tree (as it did in previous releases). The purpose of this is that if you copy the Catalyst folder to your own local user directory, you can build Catalyst locally and do not need to have write permission for the Catalina installation tree. Note that the **build_all** script already supported local builds, it was only the **build_catalyst** script that did not.
8. The **build_utilities** script now detects whether it is being run in the current directory. If so, it builds the utilities in this directory, otherwise it builds them in the utilities folder in the Catalina installation tree (as it did in previous releases). The purpose of this is that if you copy the utilities folder to your own local user directory, you can build the utilities locally and do not need to have write permission for the Catalina installation tree. Also, the build_utilities script has been modified to prompt whether the utilities binaries should also be copied to Catalina's bin directory, or simply left in the current directory. Copying the utilities to Catalina's bin directory is convenient since payload looks for them there if it does not find them in the current directory, but it requires write permission to the Catalina installation tree. If you do not have this permission, you can now copy the utilities folder to your own user directory, build the utilities there, then copy the binaries to each directory from which programs will be loaded – which may actually be a better solution if you have multiple Propeller platforms or configurations which need different versions of the utilities. Note that the utilities are required only for Propeller 1 platforms – there are currently none needed for any of the supported Propeller 2 platforms.
9. The **build_all** script in the utilities folder has been removed (it was deprecated quite a few releases ago). Use the build_utilities script instead.

The main difference is that build_all used to build binaries for all CPUs of multi-CPU platforms (such as the **TRIBLADEPROP**) whereas the **build_utilities** script has to be re-run for each CPU.

Other Changes

1. Implemented a workaround for an OpenSpin/Spinnaker bug that meant compiling Spin programs with too many short symbol names could fail unexpectedly. The current workaround is to simply use longer symbol names. Affected only Propeller 1 programs that used the Catalina Optimizer (which may generate many additional symbol names).
2. Fixed a typo in the DOSFS demo program that prevented it from compiling.
3. Updated the version of Lua used in **payload** and **blackbox** to Lua 5.4.4, which is the current version. Lua 5.1.5 is still included as a Catalyst demo program, but is no longer compiled by default (Lua 5.4.4 is now used everywhere) and Lua 5.1.5 is now deprecated and may be removed from a future release.
4. Since Lua is now compiled in COMPACT mode by default, the pre-compiled P2 demo versions of Catalyst (in *P2_EVAL.ZIP* and *P2_VGA.ZIP*) are now compiled in NATIVE mode, which improves execution speed, but at the expense of larger executables for some of the demo programs. If this is a problem, simply recompile Catalyst, specifying **COMPACT** mode as one of the parameters to the **build_all** script.
1. 5. Updated the notes about compiling Catalyst – a memory model should only be specified as an argument to the **build_all** script when compiling for the Propeller 2, not for the Propeller 1.

Release 5.2

New Functionality

1. Added a new inline pasm demo ("test_inline_pasm_5.c") to demonstrate how to CALL an inline PASM function from within an inline PASM function in a COMPACT program.
2. Updated the validation suite. Now logs more details to make it easier to identify what failed, and also added "short" scripts to do faster validation.
3. Updated the file 'globbing' demo (in folder *demos\globbing*). Tidied it up, fixed a few bugs and it now uses the correct DOS filename syntax, prints the filenames properly, and is now correctly case insensitive in all cases. The listDir() function has been re-written to be an instance of a new generic doDir() function, which calls a specified function on each matching filename.
4. Wildcard matching ('globbing') has been added to many of the Catalyst utilities. You can now specify wildcard expressions like ***.bin** or **[a-g]*.b??** - see *glob.c* for details. The options and syntax of the utilities have not changed, except for **ls** (see 5 below).

   The commands affected are:

   ls – list files and/or directories

   rm – delete files or directories

   cp – copy files

mv – move files

cat – concatenate and print files

For example, you can now say things like:

ls *.bin *.dat

mv [a-f]*.bin bin

rm ???.dat

cat *.txt

Note that wildcards can only be used in the file name part of a path, not in the directory part, so you cannot specify an argument like **/b??/*.*** expecting it to match **bin/*.***

For arguments that may be interpreted as files or directories, adding a trailing **/** ensures they will be treated as directories. For example:

ls bin -- list just the entry "bin" (if it exists)

ls bin/ -- list the contents of directory "bin" (if it exists)

Note that the built-in **dir** command does not accept wildcards because there is not enough space to do so. The same is true of the **cat** command when it is compiled as a Catalyst build-in command (it is normally compiled as an external command).

5. The Catalyst **ls** command now uses a short listing format (similar to that used by Unix or the built-in **dir** command) by default. The previous default listing format can be specified using the **-l** option, and the previous long listing format (which used to be specified by **-l**) must now be specified by **-l -l** or **-ll**. Also fixed interactive mode, which was not working in some cases.
6. 6. The Catalyst **cp**, **mv** and **rm** commands now accept **a** or **A** (for "all") at the interactive prompt, to indicate that the command should assume "yes" for all subsequent files.
7. The catalyst **build_all** script now does **clean_all** before building, to ensure there are no binaries left from previous builds.

Other Changes

1. Updated the file *catalina_compact.inc* in the target and target_p2 directories to add more notes about writing inline PASM in COMPACT programs.
2. The tiny library had a version of the C stdio function gets(). This prevented programs that used gets() from linking with the tiny library, because it ended up being multiply defined. The version in the tiny library has been renamed tiny_gets() in this library, but like the other tiny functions (i.e. tiny_printf() etc) it will be used automatically if a program is linked with the tiny library. Affected the Propeller 1 and Propeller 2.
3. Fixed a bug in the Propeller 1 SD Card plugin – multiple block writes must not be used on platforms that must disable the SD Card to use XMM RAM (e.g. because they share pins). Affected the Propeller 1 only.
4. Fixed a bug in Catalina's "unmanaged" file close function which meant that only a limited number of files could ever be opened. Did not affect the normal stdio open/close functions, but it affected the Catalyst **cat** command, which uses the unmanaged versions of open/close to save space.

5. Catalyst's example basic files are now all Unix format, not DOS format. This makes no difference to their functionality, but makes them easier to edit usng **vi** or list using **cat**.
6. Fixed a bug in DOSFS which affected the ability to delete files with zero length (e.g. using the Catalyst **rm** command) - doing so may have corrupted the file system.

Release 5.1.2

New Functionality

1. The path that Lua uses to search for modules has changed. The old path was the one used on Windows, which was not appropriate for the Propeller. The path is now:

   ?.lua;?/init.lua;lua/?.lua;lua/?/init.lua

   This means that (for example) if you say **require 'm'** then Lua will search for *m.lua*, *m/init.lua*, *lua/m.lua* and *lua/m/init.lua* in that order.
2. The Lua demos *sst.lua* and *sst-tos.lua* now use **propeller.sbrk()** instead of **threads.sbrk()**, which means the **mem** command (which displays the top of the C heap) now works when the program is run under both **lua** and **mlua**.

Other Changes

1. Fixed a bug in the threads library, which prevented some multithreaded programs from compiling. Affected only Propeller 1 programs which were compiled in **LMM TINY** mode.
2. Eliminated the file *minit.c* from Lua. This was a temporary replacement for the Lua file *linit.c* which included the threads module. Now *linit.c* will include the "threads" module if **LUA_THREADS** is defined in the file *luaconf.h*. For completeness, there is also a corresponding **LUA_PROPELLER** that is used to specify that the "propeller" module be included. This is more in keeping with the way Lua normally defines compile-time options, eliminates the need to have the extra file, and it also means that when compiling the executables (e.g. **lua** and **mlua** or **luax** and **mluax**) under Catalina, Catalina can automatically detect the compile options and include the correct modules. But if you compile **mlua** or **mluax** using gcc and posix threads, you will now need to explicitly define **LUA_THREADS** in *luaconf.h* to include the "threads" module.
3. Fixed a bug that meant the Lua "propeller" module would not compile using any compiler other than Catalina. Although it is only functional when compiled using Catalina, it should still have compiled using other compilers (such as gcc).
4. The number of threads the *test_maximum_threads.c* demo program creates has been reduced to 145, since at 150 there are some plugin combinations that do not have enough free Hub RAM to support 150 threads. Affected the Propeller 1 only.
5. The **build_all** scripts and Makefiles now always build Lua in **COMPACT** mode on the Propeller 2, because **NATIVE** mode does not leave enough Hub RAM for the various demo programs.

Release 5.1.1

New Functionality

1. Added a new demo folder (*xeprom*) containing programs that demonstrate how to read from the EEPROM from XEPROM XMM programs. In such programs, the EEPROM cannot be read independently (e.g. using an I2C driver) because it is constantly in use by the cache. This applies only to the Propeller 1, and requires a platform with an EEPROM larger than 32k, such as a FLIP, a C3 or a HYDRA.

Other Changes

1. Fixed a bug that meant the **-e** command line switch to Catalina, which is used to specify that a **.eeprom** file be generated instead of a **.binary** file did not work correctly in some instances. In particular, it could not be used in conjunction with the **-C XEPROM** option. This affected the Propeller 1 only, and only Catalina version 4.9.3 or later.
2. The **catalina_clean** utility script had not been updated recently, and would miss cleaning some directories when the optional **all** parameter was specified.

Release 5.1

New Functionality

1. Decoding the **CATALINA_DEFINE** environment variable was taking place AFTER the symbols to specify the clock frequency were defined, so the Catalina symbols **MHZ_220**, **MHZ_260** & **MHZ_300** only worked when specified on the command line, and had no effect if specified using **CATALINA_DEFINE**.
2. Added **MHZ_200** as a Catalina symbol, to make it easier to build Catalyst using 200Mhz on the Propeller 2 – this is now the recommended frequency for building Catalyst on the P2, to work around an obscure bug that occurs at 180Mhz. Note that the Propeller 2 reference manual, the tutorial Getting Started with Catalina and the documentation in the Propeller 2 VGA plugin source file all mistakenly referred to this Catalina symbol when they should have referred to the existing **MHZ_220** symbol. All these documents have now been updated.
3. The Lua "propeller" module has had the functions sleep, **msleep** and **sbrk** added. These functions are the same as the ones in the "threads" module, but are handy when the propeller module is used in a non-threaded Lua program.
4. The Lua "threads" and "propeller" modules now accepts a string parameter. If "lua" is specified, these functions return the version of Lua. Otherwise they return the version of the module.
5. Lua versions of the classic Star Trek game (*sst.lua* and *sst-tos.lua*) are now included as Lua test programs.
6. A change was made to the **gettoken** procedure in Dumbo Basic 1.0, which was introduced in Catalina release 4.7. This change led to a syntax error when executing *TREK15.BAS*. This change has been reverted in Dumbo

Basic 1.1, but if this breaks any existing basic code, it can be re-implemented by modifying the file *basic.c* and changing the #define of **NEW_GETTOKEN** to 1.

7. When compiled for the Propeller, Dumbo Basic now has a delay on exit of 100ms to allow time for any pending output to be printed. This prevents error messages or final program output from being truncated.

Other Changes

1. The command-line options to set the clock frequency (**-f**, **-F** and **-E**) were not described in the documentation. These options apply to the Propeller 2 only.
2. On the Propeller 2, it is now recommended that Catalyst be compiled using a 200Mhz clock speed. This is a work-around for an obscure bug which only seems to affect some programs loaded from the SD card and which use a clock speed of 180Mhz.
3. A bug in the SD Card plugin was preventing programs being able to write to some SD cards (reading was ok). This affected the Propeller 2 only.

Release 5.0.3

New Functionality

1. The default version of Lua (e.g. built by the Catalyst build_all scripts) is now Lua 5.4.4. However, Lua 5.1.5 is still included and can be built manually if required.
2. The Lua threads module has a new **version()** function which returns the **LUA_VERSION_NUM** associated with the Lua version. The main use for this is to allow for changes made in Lua between versions, such as the changes to the garbage collection options in various Lua releases.

   The possible return values are:

   501 – Lua Version 5.1.x

   502 – Lua version 5.3.x

   503 – Lua version 5.3.x

   504 – Lua version 5.4.x

Other Changes

1. From version 5.2 onwards, Lua no longer "requires" the coroutines module by default, so the Lua threads module now does this explicitly.
2. Some functions in the threads module were returning floating point values when the value was really an integer. Lua versions 5.2 and earlier only supported floating point values, so this was the only option. From version 5.3 onwards, Lua supports integers as well as floating point values, and in some cases it is more appropriate for the function to return an integer.
3. For consistency with the **threads.factories()** function, **threads.workers()** now returns the current number of workers. However, note that if this function is used to CHANGE the number of workers then the value returned may be anywhere between the old number of workers and the new number. This function merely requests a new number of workers – actually creating or destroys the workers may not happen immediately.

4. The Makefile for Lua-5.4.4 had not been updated to build Multi-processing Lua on Windows or Linux.
5. The document **Lua on the Propeller 2 with Catalina** has been updated to note a limitation of using **coroutine.yield()**, or the **put** and **get** message passing functions – i.e. that you cannot hold a mutex locked while you use these functions

Release 5.0.2

New Functionality

This release contains no new functionality.

Other Changes

1. Fixed a bug in the Lua threads module when sending and receiving messages. The program could lock up under certain circumstances. Since the Lua threads module is only supported on the Propeller 2, this affected the Propeller 2 only.
2. Multi-threaded Lua examples 4 has been updated to eliminate a race condition.
3. Multi-threaded Lua example 8 has been updated to make it more evident the example is using preemptive multi-tasking.
4. Multi-threaded Lua examples 11 and 12 have been added to demonstrate interactions between co-routines and threads.
5. The document **Lua on the Propeller 2 with Catalina** has been updated to reflect the above changes. Also, some terminology issues have been clarified.

Release 5.0.1

New Functionality

6. Modified various Catalyst commands to allow combined command-line options – e.g. **-dh** now means the same as **-d -h**

Other Changes

1. Fixed a bug in the DOSFS file system which allowed a file to be opened as a directory using the **DFS_OPENDIR** option to **DFS_OpenFile** instead of returning that a directory of that name did not exist. This confused the Catalyst cp command when copying files into a directory.
2. Fixed a bug in the Catalyst rm command that prevented recursive file deletions.
3. Fixed a bug in the Lua threads module that prevented the number of factories being reduced. Also, the Lua threads module documentation has been updated to reflect the fact that reducing the number of factories now necessarily terminates all workers first (they are restarted after the number of factories has been reduced) and that this should therefore only be done from the main thread and when that thread is executing on factory 1.
4. Modified Lua to zero all newly allocated blocks of memory. Lua seems to assume all new blocks of memory will be initialized to zero, which is not

actually guaranteed by C, so this led to some Lua programs behaving unexpectedly. While this fix resolves the issue, it does so at a small run-time cost, so further investigation is required. Multi-threaded Lua example 9 was badly affected by this issue and has been updated. The Lua threads module documentation has also been updated.

Release 5.0

New Functionality

5. The standard C memory management functions (i.e. malloc, realloc, calloc, free) and a new system break function (sbrk) are now all thread safe. For details, see the "Multi-processing, Locks and Memory Management" section in the **Catalina Reference Manual**.
6. Catalina's dynamic memory management has been modified to make it less prone to running out of memory when lots of small randomly sized chunks of memory are allocated, freed or re-allocated. For details, see the "Multi-processing, Locks and Memory Management" section in the **Catalina Reference Manual**.
7. A new function has been added that can be used to assign one or more locks to protect services offered by plugins in multi-cog or multi-thread programs. For details, see the "Multi-processing, Locks and Memory Management" section in the **Catalina Reference Manual**.
8. An example of a public domain itoa function has been added to the folder demos\examples – it is called ex_itoa.c. The itoa function is also used in the new demo program test_thread_malloc.c (as an alternative to using sprintf, which can make the program too large for the Propeller 1).
9. Catalina used to use 4 bits for the lock in the service registry. The Propeller 1 only had 8 locks, so this was sufficient to represent all possible locks plus a bit to indicate "no lock". But the Propeller 2 has 16 locks, so Catalina must now use 5 bits for the lock. To make space for the extra bit, the number of bits used to hold the cog-specific service code has been reduced from 8 to 7. This still allows up to 127 services, but less than 40 such services are implemented even by the most complex plugins (the HMI plugins), so this is unlikely to be a problem. If it becomes so, the size of each service entry can simply be increased from a 16 bit word to a 32 bit long.

   Although this change is only required for the Propeller 2, to keep the library functions consistent, it has been made on the Propeller 1 as well.
10. The speed of the p2asm assembler has been improved – it is now approximately twice as fast. I am still not sure why it is so slow on Windows compared to Linux, where exactly the same code executes about 10 times faster on the same machine.
11. If a baud rate is not specified and a file with a ".bin" extension is loaded, payload now assumes it is loading a Propeller 2 and sets the baud rate to 230400. This can be overridden by specifying a baud rate either on the command line or via the **PAYLOAD_BAUD** environment variable.
12. Windows 10 has a bug in the Command-Line interpreter. When loading a program using payload, it crashes (taking payload down with it) if the Wndows option "Wrap Text Output on Resize" option is not set in the Layout

Options (in the Properties dialog box). This can leave a payload task executing in the background which needs to be terminated manually (e.g. using Task Manager) in order to release the USB port.

13. The translation of outgoing CR to LF has been disabled in the payload interactive terminal mode by default, but mode 16 has been added to restore the original behaviour if needed. The translation was preventing the use of payload to interact with TAQOZ and the P2 Monitor - the correct mode to use for these is now -q1.

    For instance, if you have a Propeller 2 connected to port X, you can communicate with TAQOZ by entering a command like:

    ```
    payload -i -q1 -pX
    ```

    Then reset the Propeller 2 and enter the usual TAQOZ entry sequence (i.e. **'>' SPACE ESC**) or P2 monitor entry sequence (**'>' SPACE CTRL-D**)
14. Catalina now supports a "thin" binding to Posix threads. See the "Posix threads (pthreads) support" section in the Catalina Reference Manual.
15. Lua module "threads" adds multi-threading capabilities to Lua (for the Propeller 2 only). The document **Lua on the Propeller 2 with Catalina** describes this new functionality in detail.
16. The interactive versions of Lua (and Multi-threaded Lua) now support trivial line editing (e.g. backspace is now recognized).
17. Catalina now supports setpin(), getpin() and togglepin() for both the Propeller 1 and 2. These are defined in propeller.h:

    ```
    setpin(pin, value) - sets pin to output and sets value
    setpin(pin)        - sets pin to input and gets value
    togglepin(pin)     - sets pin to input and toggles value
    ```

    These functions are also provided in the Lua propeller module, described in the document **Lua on the Propeller 2 with Catalina**.

Other Changes

1. Support for the dual-CPU **MORPHEUS** platform has been dropped. The Morpheus was supported up to Catalina 4.9.6. The Morpheus required special complex build scripts, and some of the large demo programs would not run because of the unique and expensive (in terms of cogs) VGA driver it required. Also, many of the demo programs required both CPUs to run at all since the keyboard and mouse was on one CPU but the VGA hardware was on the other. This limited the usefulness of having dual CPUs. The Morpheus-specific scripts and drivers from that release may contine to work in subsequent releases, but installing, testing and maintaining them is now up to the user. They will no longer be routinely included as part of each Catalina release.

   Support for the simpler three-CPU **TRIBLADEPROP** will remain, primarily as an example of how to support multi-CPU Propeller systems, which typically require the use of multi-stage loaders and proxy HMI drivers.
2. Changes to the Catalina Optimizer to allow it to recognize more code that can be optimized. Generally only affected hand-coded PASM, where the spacing in the code did not match exactly what the Catalina code generator would produce (e.g. if you used tabs as a separator instead of spaces).
3. Make it more explicit in the documentation that the integer-only version of

the C standard library (libci) does not include support for sprintf() or vsprintf() - if these functions are required, use one of the other libraries (e.g. libc). This is to keep libci as small as possible.

4. Fixed some instances where the C library stdio was not re-entrant, and hence not thread safe or multi-cog safe.
5. The Catalina HMI function t_float() was not re-entrant, and hence not thread safe or multi-cog safe.
6. Use dynamic io buffers for stdio (previously, Catalina used static io buffers which was more efficient but not multi-cog safe or thread safe).
7. Minor changes to the include files *propeller.h*, *propeller2.h* and *catalina_cog.h* – it is now possible to include any or all of them, and in any order on both the Propeller 1 and 2 and they should not interfere. (but of course, *propeller2.h* defines functions that may not be available on the Propeller 1, and vice-versa). There have been some changes to the function definitions in these files (not to the functions themselves) that could conceivably cause issues for programs that relied on specific definitions.
8. There was an error in the *Makefile* in the *demos\p2* directory that prevented some of the demo programs building.
9. The reference manual documentation for _lockset (etc) has been updated, as both the Propeller 1 and 2 reference manuals had gotten out of sync with the actual software.
10. The various reference manual sections dealing with Multi-Processor support (i.e. multi-cog, multi-threading and multi-model) have been combined and extended.
11. Fixed a bug in DOSFS that led to the Catalyst "rm" command thinking a file to be deleted was a directory when in fact it simply didn't exist.
12. Fixed a bug in xvi (aka vi) which meant that if a yank or delete command was the first thing attempted, it may have given a spurious "Not enough memory to perform delete" error.
13. Fixed an issue in some of the build_all scripts that would not detect a Propeller 2 if it was specified using **CATALINA_DEFINE** instead of on the command line. Now, a P2 is assumed if the first 2 characters of the first parameter to the build_all script are "P2" OR the first two characters in the **CATALINA_DEFINE** environment variable are "P2"

Release 4.9.6

New Functionality

14. Updated the payload loader to allow EOT (Ctrl-D) to be passed to the application program in interactive mode without also closing the terminal emulator. Now pressing Ctrl-D ONCE passes an EOT to the application but does not terminate payload. To terminate payload you must now enter TWO SUCCESSIVE Ctrl-D characters.
15. Updated the minimalist Lua demo program (*min.c*) so that it would compile under both Lua 5.1.5 and Lua 5.4.3 (which changed the names of some Lua library functions).
16. Updated the Lua 5.1.5 and 5.4.3 Makefiles to include pattern templates. This simplifies the compilation of Lua programs like *min.c* - you can now use

just the Catalina Makefile to build Lua programs. For example, to build the minimalist demo:

```
make -f Makefile.Catalina clean min.bin
```

17. Added the Lua execution engine (**luax**) that executes only compiled Lua programs to Lua-5.4.3. This saves Hub RAM by not loading the Lua parser if all the Lua programs to be executed are pre-compiled.
18. Added **clean_all** scripts to the Catalyst demo folders, to make it easier to remove compiled versions of all the Catalyst programs.
19. Added **clean_all** scripts to the Catalina source folders, to make it easier to remove compiled versions of all the Catalina programs.
20. Removed the -C NATIVE from the Catalyst build_all and Makefiles, and some of the -O5 options on individual demo programs. This means that you can now specify OPTIMIZE on the command line if you want Catalyst programs to be optimized, and on the Propeller 2 you can now choose to compile all the demo programs as COMPACT (instead of NATIVE). Note that this applies to the Propeller 2 only. Do not specify a memory model on the Propeller 1 – the core programs must always compiled as COMPACT on the Propeller 1 because otherwise they would be too large for a Propeller 1 without XMM RAM, and the other programs must usually be compiled in LARGE mode.

Other Changes

1. Fixed some errors in Dumbo Basic that stopped it compiling in COMPACT mode. Affected the Propeller 2 only.
2. Release 4.9.5 fixed one bug in the COMPACT mode optimizer, but introduced another, which has been fixed in this release. Affected the Propeller 2 only (and the only program known to be affected was the vi text editor, which would not compile in COMPACT mode)

Release 4.9.5

New Functionality

1. Updated Lua to version 5.1.5 in both payload (where it is used for scripting) and Catalyst (where it is used as a demo program). Version 5.1.5 as the last release of Lua 5.1. On a Propeller 1 it must be compiled in LARGE mode. On a Propeller 2 it can be compiled in either NATIVE or COMPACT mode.

   Catalyst will continue to incorporate Lua 5.1 because it is small enough to be useful on a micro controller like the Propeller even when compiled in NATIVE mode (for speed). Also, Lua 5.1.5 incorporates a new option to execute compiled Lua programs using a Lua execution engine that does not load the Lua parser (**luax**). This saves even more Hub RAM when all the Lua code to be executed is pre-compiled using the Lua compiler (**luac**).

   For example, to execute the *life.lua* demo program, you can now either do:

   ```
   lua life.lua
   ```

   or

   ```
   luac -o life.out life.lua
   luax life.out
   ```

   Lua version 5.4.3 is also included as a Catalyst demo, but is not compiled by

default. Version 5.4.3 is the current release of Lua 5.4. It should be compiled with the COMPACT option, or it may be too large to be much use. For example:

```
build_all P2_EVAL SIMPLE VT100 COMPACT
```

On the Propeller 2, the code sizes of Lua 5.1 vs Lua 5.4 are:

	NATIVE	COMPACT
Lua 5.1 (no parser)	139k	250k
Lua 5.1 (with parser)	166k	297k
Lua 5.4 (with parser)	227k	412k

Hence Lua 5.1 is quite usable in both NATIVE and COMPACT modes, and with or without loading the parser, whereas Lua 5.4 is really only usable in COMPACT mode. This is because Lua 5.4 itself takes so much Hub RAM that there is not much left to run anything except small Lua programs (the above sizes are the code sizes only – there is also stack and heap space required to actually execute a Lua program). Note that there is currently no option to execute Lua 5.4 programs without loading the Lua parser. This option may be added in a later Catalina release.

Note also that on memory constrained systems such as a Propeller, the Lua garbage collector sometimes requires a "tweak" to ensure it runs often enough to stop Lua programs running out of memory. The *life.lua* demo program is an example – it does many string concatenations per generation, and if the garbage collection does not run often enough, the program can run out of memory. To prevent this, the following line has been added to the program:

```
collectgarbage("setstepmul", 500)
```

Refer to the Lua documentation for more details.

2. Updated **MAX_SYMBOLS** in p2asm to be 20,000 (was 10,000) - this is required to allow Lua 5.4.3 to be compiled with optimization enabled.
3. Added decoding of Catalina symbols **MHZ_300** and **OPTIMIZE**. This allows these options to be specified on the command line of the build_all scripts, or via CATALINA_DEFINE:

   MHZ_300 set the default Propeller 2 clock to 300Mhz. Note that this may not work on all Propeller 2 platforms. USE THIS OPTION WITH CAUTION, BECAUSE IT MAY NOT BE IMMEDIATELY APPARENT THAT A PROGRAM IS NOT EXECUTING CORRECTLY!

   OPTIMIZE set the optimizer level to 5.

   For example, you can now say:

   ```
   build_all P2_EVAL MHZ_300 OPTIMIZE
   ```

   or

   ```
   set CATALINA_DEFINE=P2_EVAL MHZ_300 OPTIMIZE
   build_all
   ```

   These symbols are not intended to be used on the command line, but they can be used via the -C option. For example:

   ```
   catalina hello_world.c -p2 -lci -C MHZ_300 -C OPTIMIZE
   ```

   would be the same as:

   ```
   catalina hello_world.c -p2 -lci -f300Mhz -O5
   ```

Other Changes

1. Reformatted some Catalyst text files (e.g. Catalyst.txt) to be Unix format instead of DOS. This makes them easier to view in the Catalyst text editor (vi).
2. Fixed a bug in the Catalina Optimizer when optimizing COMPACT programs. This may have resulted in programs that would not execute correctly. Affected only Propeller 2 COMPACT programs.
3. Fixed a bug when compiling Catalyst – it would not compile on the Propeller 2 if the file *sio_array.h* was not present in the core folder, even though that file is only required when compiling for the Propeller 1.

Release 4.9.4

New Functionality

1. Added a new serial HMI option called **SIMPLE** (supported on the P2 only). This option uses P2 smartpins to implement a very simple serial interface. The advantage of the new HMI option is that it saves a cog over the default serial HMI option (**TTY**). The disadvantage is that may not support very high baud rates – but baud rates up to 230400 are supported, which means it is sufficient for a serial terminal. To use it, define the symbol **SIMPLE** on the command line. For example:

```
catalina -p2 -lci hello_world.c -C SIMPLE
payload -i -b230400 hello_world.bin
```

2. The serial baud rate can now be specified on the command line, either by using the **-B** command line option, or by defining the symbol **_BAUDRATE** using a complex definition such as **-C "_BAUDRATE=xxx"**. This is supported on the Propeller 2 only. The payload loader now also accepts **-B** to specify the baud rate – i.e. payload now accepts either **-b** or **-B** (whereas catalina only accepts **-B** since **-b** already meant something else). For example:

```
catalina -p2 -lci -B9600 hello_world.c
payload -i -B9600 hello_world.bin
```

or

```
catalina -p2 -lci -C "_BAUDRATE=9600" hello_world.c
payload -i -B9600 hello_world.bin
```

If it is not specified on the command line, the default value defined for the platform (in *Catalina_platforms.inc* in the *target_p2* directory) will be used.

Other Changes

1. Complex symbols had not been implemented in the Catalina Optimizer, which meant that the optimizer could not be used if the command line contained a complex definition such as:

```
-C "name=value"
```

2. Payload now allows baud rates down to 300 baud. There was really no reason payload ever limited the baud rates (it used to impose a minimum of 19200 baud) except that at the lower baud rates the program load times become so long that it was not much use. However, this was before payload could also be used as a serial terminal emulator, and in some cases low baud rates are useful for testing. However, note that the P2 smart pin UARTs

use a 16-bit value for baud timing which can limit baud rates – to use baud rates lower than 4800 baud you may need to also specify a lower system clock frequency. For instance, to use a baud rate of 300 baud, you may need to use a clock speed of 12Mhz or lower:

```
catalina -p2 -lci hello_world.c -C SIMPLE -B300 -f12Mhz
```

3. There was a bug in the Catalina HMI **t_bin()** function, which prints the 32 bit binary representation of its integer or unsigned argument. The value was effectively being shifted left by 3 bits.

Release 4.9.3

New Functionality

1. You can now define complex Catalina symbol definitions on the command line to assign a symbol a value (i.e. in addition to simply defining the symbol itself). This is currently supported only on the Propeller 2 – the Propeller 1 still only supports simply defining such symbols. You must put the complex symbol definition in double quotes, such as:

   ```
   -C "name=value"
   ```

   Note that defining complex C symbols was already supported using similar syntax (but using the -D command-line option).

2. You can now request a specific clock frequency, or specify arbitrary clock parameters on the command line. This is normally done using the new **-f**, **-F** and **-E** command line options. This is supported on the Propeller 2 only – on the Propeller 1 you must still set the initial clock parameters in the appropriate *<platform>_DEF.inc* file (e.g. *C3_DEF.inc* for the C3 platform).

   The meaning of the new command-line options are as follows:

   -f requested frequency for which to calculate clock parameters

   -F xtal (XI) frequency to use in frequency calculation (default 20Mhz)

   -E error limit for frequency calculation (default 100khz)

   In most cases, you just use the **-f** option by itself. Note that **m** and **k** suffix chars (case insensitive) are supported by these options, and mean to multiply the value specified by 1,000,000 or 1,000 respectively. This means you can say things like:

   ```
   -f 260Mhz
   ```

   or

   ```
   -f 123456kHz
   ```

   for example:

   ```
   catalina hello_world.c -p2 -lc -f300Mhz
   ```

   Specifying a frequency via **-f** causes Catalina to calculate the clock parameters required to achieve it, and defines and uses three Catalina symbols if it is possible to achieve the specified frequency within the error limit. These symbols are:

   _CLOCK_XDIV : XI divider (1..64)

   _CLOCK_MULT : XI multiplier (1..1024)

   _CLOCK_DIVP : VCO divider (1, or even numbers from 2 to 30)

   These can also be defined manually, but they will only be used if all three are

defined. The following clock parameters can also be defined:

_CLOCK_XTAL : XI frequency
_CLOCK_OSC : 0=OFF, 1=OSC, 2=15pF, 3=30pF
_CLOCK_SEL : 0=rcfast, 1=rcslow, 2=XI, 3=PLL
_CLOCK_PLL : 0=PLL off, 1=PLL on

To manually specify these parameters, use complex symbol definitions, which means you need to enclose them in double quotes – e.g:

```
-C "_CLOCK_SEL=2"
```

or

```
-C "_CLOCK_XTAL=25000000"
```

or

```
-C "_CLOCK_XDIV=1" -C "_CLOCK_MULT=9" -C "_CLOCK_DIVP=1"
```

Note that the following two command-line options have the same effect:

```
-C "_CLOCK_XTAL=25000000"
```

and

```
-F 25Mhz
```

but that the "25Mhz" notation is not supported by the -C option. This means you CANNOT also say:

```
-C "_CLOCK_XTAL=25Mhz"    <-- WRONG!
```

The symbols **MHZ_260** and **MHZ_220** remain supported, but are now translated internally into **-f 260MHz** and -**f 220MHz** options, which will have the same effect.

If conflicting clock parameters are specified, the last one on the command line will be used. If no clock parameters are specified, or the requested frequency cannot be achieved, the default clock parameters defined for the platform in *catalina_platforms.inc* will be used.

3. The Catalina parallelizer has had a bug fixed by adding new options that allow better control of the locks used to control access to exclusive code segments – they can now be local (i.e. local to the file in which they are used) or global (i.e. used across multiple files). This is done by adding a **lock** and **extern** option to the exclusive and shared pragmas (this is demonstrated in a slightly updated test_6 in the *demos\parallelize* directory). See the document **Parallel Processing with Catalina** for more details.

Other Changes

1. Improved the timing of interrupts and the use of the ticks value in the NATIVE version of the multi-threaded kernel for the Propeller 2. Now, interrupts occur about every millisecond irrespective of the value of the ticks value of the thread. This makes threaded programs more responsive, timing operations more accurate, and means that thread changes take effect sooner – for instance, changing the ticks value will take effect after about a millisecond, not at the next context switch, which may only occur several seconds later.
2. The **_lockclr()** function now correctly returns the previous state of the lock on the Propeller 2 (it already did so on the Propeller 1).

3. Fixed a bug in *test_inline_pasm_2.c* demo program which meant it would not work in COMPACT mode.
4. Fixed bugs in various thread functions, which may have led to multithreaded programs deadlocking on the Propeller 2.

1. Various minor improvements to the multithread demo programs.

Release 4.9.2

New Functionality

1. Added **_waitsec()**, **_waitms()** and **_waitus()** to the Propeller 1 and Propeller 2 libraries. Added **_waitx()** to the Propeller 1 library (as a synonym for **WAIT()**. It already existed in the Propeller 2 library.
2. Added **_pinstart()** and **_pinclear()** to the Propeller 2 libraries.
3. Added a new include file (*spin2cpp.h*), which is included by *propeller.h* and *propeller2.h* if the symbol **__SPIN2CPP__** is defined. This allows for symbols assumed to exist by the **spin2cpp** utility to be defined if they don't already exist (e.g. due to name differences between C compilers).
4. The default for the CC field of the clock mode setting for the P2_EVAL, P2_CUSTOM and default P2 platform is now %10 (15pf) and not %01 (OSC). It is still %01 for the P2_EDGE.
5. Updated Catalina's version of p2asm to version 0.018. Minor bug fixes and inclusion of "encod" keyword. Thanks to Dave Hein.

Other Changes

1. Updated the Makefiles in the demos folder to add -p2 to the command line options if making a file with a **.bin** extension.

Release 4.9.1

New Functionality

None. Release 4.9.1 is a bug fix release for release 4.9.

Other Changes

5. Catalina Geany has been updated to fix a crash on startup under Windows. This affected both Propeller 1 and Propeller 2 programs. Linux versions of Catalina were not affected.
6. A bug in Catalyst support for SMALL FLASH binaries has been fixed. In previous releases only FLASH programs with the LARGE memory layout would load correctly under Catalyst. Now Catalyst will correctly load FLASH programs compiled with the SMALL memory layout as well. Note that old program binaries will not load correctly – they will need to be recompiled with Catalina 4.9 to do so. This affected Propeller 1 programs only.
7. A bug in the PROPTERMINAL HMI support has been fixed. This affected Propeller 1 programs only.
8. The project file chimaera.geany specified the -p2 flag in the project's Catalina Options field. This meant the project would not compile correctly for the Propeller 1 even if a Propeller 1 platform was specified (e.g. in the

CATALINA_DEFINE environment variable).

9. If the LCCDIR environment variable is not set, Catalina now assumes it is installed in *C:\Program Files (x86)\Catalina* (it used to assume it was in *C:\Program Files\Catalina*), which confused lcc.

Release 4.9

New Functionality

1. There has been a major rewrite of all the demo programs and build scripts in the demos directory and all its subdirectories. Makefiles are now provided for all the demo programs, which can be used either manually (i.e. by invoking **make** on the command line) or via the Catalina Geany IDE. All platforms are supported by these new Makefiles except for the MORPHEUS and TRIBLADEPROP platforms (which still use the old batch scripts). To use the Makefiles on Windows, this means you need to have a version of **make** and some core utilities that it needs installed. The Windows installer can install these for you if you do not already have them (e.g you don't already have MinGW installed), or you can download them from here:

   ```
   http://gnuwin32.sourceforge.net/packages/make.htm
   http://gnuwin32.sourceforge.net/packages/coreutils.htm
   ```

   The recommended version of make to use with Catalina is 3.81. The recommended version of the core utilities is 5.3.0.

   Note that Catalina can still be used as a command-line compiler without installing **make**, and even Catalina Geany can still be used without it to compile simple programs. But to build more complex examples (such as the Catalyst demos) or use the various build_all scripts, a version of **make** is required.

   Most of the 'build_all' scripts have all been rewritten to just invoke **make** internally, but other than that should work similarly to how they worked in previous releases. So there are now four different methods you can use to make the demo programs and your own C programs.

   For instance, in the demos directory, there are now four ways to compile the *hello_world.c* program ...

   a) Manually, by invoking catalina yourself on the command line with appropriate command-line options – e.g:

      ```
      catalina hello_world.c -lci
      ```

      You can define Catalina symbols on the command line to specify options for your compilation, or else use the CATALINA_DEFINE environment variable. For example, the following will give identical results:

      ```
      catalina hello_world.c -lci -C C3 -C COMPACT -C HIRES_VGA
      ```

      and (on Windows)

      ```
      set CATALINA_DEFINE=C3 COMPACT HIRES_VGA
      catalina hello_world.c -lci
      ```

      or (on Linux)

      ```
      export CATALINA_DEFINE="C3 COMPACT HIRES_VGA"
      catalina hello_world.c -lci
      ```

      The advantage of using the CATALINA_DEFINE environment variable is that if you are working with a single Propeller platform, you don't

need to specify this in every compilation command. Note that you don't use the -C command line switch when defining symbols using CATALINA_DEFINE.

b) Using **make**. For all the Makefiles provided in the demo directory and its subdirectories, you can make either a single target, or all the targets specfied in the Makefile – e.g:

```
make hello_world
```

or

```
make all
```

Note: if you see a message like the following:

```
make: Nothing to be done for `hello_world'
```

then it means that **make** did nothing because the program binary already exists and is up to date. You can force **make** to rebuild the program either by first saying 'make clean', or by invoking **make** with the -B command line option – e.g:

```
make -B hello_world
```

Note that compiling with **make**, like all Catalina compilations, will use any Catalina symbols defined in the CATALINA_DEFINE environment variable. This allows generic Makefiles to be written more easily. All the Makefiles provided as demos use the **catalina_env** command to print out all Catalina environment variables before they build the programs to remind you what is specified in CATALINA_DEFINE.

c) Using the **build_all** scripts, with or without appropriate Catalina symbols specified to the command - e.g :

```
build_all
```

or

```
build_all P2_EVAL NATIVE VT100 CR_ON_LF
```

The build_all scripts are now wrappers around **make**. They don't work exactly the same way they did in previous releases, partly because the demo directories have been restructured, but they still accept command line parameters to specify the Catalina symbols to define for the build. By convention, the first parameter is always the platform, and Propeller 2 platforms should always start with "P2" (e.g. P2_EVAL, P2_EDGE, P2D2 etc).

The build scripts, like all Catalina compilations, will use any Catalina symbols defined in the CATALINA_DEFINE environment variable. Note that if you specify conflicting symbols as parameters to the build_all script, it will print a message about the conflict and not compile any programs.

d) Using Geany. You can specify a Geany project on the command line, or just start Geany with no project specified (which will start Geany with whatever project was last loaded) and then manually load the project file in Geany - e.g:

```
catalina_geany hello_world
```

or

```
catalina_geany
```

Then, in Geany you have two sets of commands on the Build menu. Note that in Geany the build menu is determined by the type of file you

have currently selected. You must have a C file (i.e. a file with a ".c" extension) open and currently selected to see the C Build commands.

The first set are the simple build commands:

Compile	compile the current file only and produce an object file.
Build	compile the current file only and link it to produce an executable.
Compile All	compile ALL the C files in the current directory to separate object files, but do not link them.
Link	link ALL the object file in the current file to produce an executable. The executable will be named for the current file.
Clean	remove all binary, object, listing and executable files in the current directory (similar to 'make clean').

These commands are suitable for C programs that consist of a single C file, or when all the C files in the directory belong to the same project and there are no dependencies between them, and no other steps required except compiling and linking.

After the simple build commands on the Build menu, you will see the two **make** build commands:

Make	Make the current project using the Makefile in the current directory – i.e. invoke **make** specifying the current project as the target.
Make (Custom Target)	Make a custom target. Geany will prompt for the target name, which can be the special targets 'clean' or 'all'.

Like all Catalina compilations, Geany will use any Catalina symbols defined in the CATALINA_DEFINE environment variable. You can set this variable to specify to your platform and memory model. For instance, on Windows you might say:

```
set CATALINA_DEFINE=P2_EVAL COMPACT
```

On Linux, the syntax to set environment variables is different. You might instead say:

```
export CATALINA_DEFINE="P2_EVAL COMPACT"
```

Then, no matter which method you use, your programs will be compiled as if you had specified -C P2_EVAL -C COMPACT on the command line. Note that if you specify conflicting symbols as parameters to the build_all script, it will print a message about the conflict and not compile any programs.

Catalina Geany has the ability to specify options to its build commands. You can specify these in the Project Properties menu item. Typically, the options would include libraries, optimization levels, and any Catalina symbols you might want to define. For instance, the options specified for "hello_world" might be as follows (multiple lines are acceptable here):

```
-p2
-lci
-O5
-C NO_REBOOT
```

This means compile for the Propeller 2, use the integer version of the standard C library, use optimization level 5, and define the Catalina symbol NO_REBOOT, which prevents the program from rebooting the Propeller on exit from the main function (which in some cases might prevent all the output from being sent to the terminal before the Propeller reboots).

These options are used by the simple build commands. In order to make them accessible to the **make** commands, the Make command actually used is as follows:

```
make "%n" CATALINA_OPTIONS="%o"
```

This allows Geany to pass the options defined for the project to the project Makefile using the variable CATALINA_OPTIONS. The Makefiles provided for all the demo programs check whether CATALINA_OPTIONS have been passed, and try and use sensible default options if not. So even if you are using **make** stand-alone, it is not usually necessary to specify options on the command line. This is what allows you to say (for example) just:

```
make hello_world
```

instead of having to say:

```
make hello_world CATALINA_OPTIONS="-lci -C NO_REBOOT"
```

Like the build_all scripts in the previous releases, all the Makefiles provided in the demos folder attempt to determine automatically if you are compiling for a Propeller 1 or 2, based on either the suffix, or (if no suffix is specified) then on whether the substring "P2" appears in any of the Catalina symbols defined in CATALINA_DEFINE or if "-p2" appears in CATALINA_OPTIONS. This means you do not generally have to explicitly add the -p2 option unless you are compiling for the default platform (and are therefore not specifying any specific platform).

The Makefiles in the demo folders contain default rules that know how to make .bin, .binary and .eeprom executables. So, for example, if hello_world.c exists in the folder, you can simply say:

```
make hello_world.bin
make hello_world.binary
make hello_world.eeprom
```

to build the executable with default options (which can also be specified in the Makefile) and get the expected result. You can use the Makefile in the main demo directory as a template for your own Makefiles.

Note that **make** uses only the file date/time to determine if the executable needs to be remade, and is not aware of any changes you may make either to CATALINA_DEFINE or to Geany build options – so in some cases you need to use 'make clean' to remove old executables, or add the -B option to 'make' to force it to rebuild the executable. Or you can specify clean as your first target – e.g:

```
make clean hello_world
```

2. A new build command option has been added to the Catalina version of Geany. Now as part of a project build command, you can specify %n which will be replaced with the name of the project. By default, this option is now added to the 'make' build command which is now:

   ```
   make "%n" CATALINA_OPTIONS="%o
   ```

   The purpose of this is to allow a single Makefile to be used to make multiple projects with sources in the same directory. This capability is used in many of the Catalina demo folders, rather than having to have a separate subdirectory (and a separate Makefile) for each individual demo program. Instead, the Makefile can simply have multiple targets, named for each project that uses the Makefile.
3. Catalina Geany has been modified to allow relative file paths. This allows geany project files to be independent of where they are installed. This means, for example, that you can more easily copy all or part of the demos directory to another location to build the example programs. You no longer need to build them in place in the Catalina program folder. Note that in Geany you can specify “./” (without the quotation marks) as your project base path in the project properties. This applies on both Linux and Windows (even though Windows doesn't normally use "/" as a path separator) - it is not really a path specifier, it is a signal to Geany to use the path to the Geany project file as the base directory for the project.
4. A new set of definitions has been added for P2_EDGE in the configuration file *catalina_platforms.inc* in the target_p2 folder. Currently, all the values defined are the same as the P2_EVAL board, but they can be edited if desired.

Other Changes

1. The source code of the Catalina Optimizer is now included in this release.
2. A bug in the Catalina Optimizer led to it missing some opportunities to optimize unnecessary assignments just before a function returned.
3. Removed the -F and -B command line options to catalina and catbind. These options were used to invoke the srecord utility to produce output in another binary format (e.g. Motorola S records or Intel Hex records). Generally, this was used for stand-alone EEPROM programmers. If this functionality is still required, then the srecord utility can be used stand-alone. See http://srecord.sourceforge.net/ for the latest version of the srecord utility.
4. Catalyst has been moved to a subfolder of the demos folder. The main purpose of this is to allow the entire demos folder to be easily copied to a users directory, allowing all the Catalina demos to be build in place without requiring administrator privileges or write permission to the Catalina program folders (this is however still required to build the Catalina platform-specific utilities).
5. The **build_utilities** batch script now warns you in advance that you need administrator privileges (or at least write permission to Catalina's *bin* directory) to install the resulting binaries in the Catalina program folder before it begins compiling the utilities (the warning used to only be given at the end).
6. A minor bug in decoding the CATALINA_INCLUDE environment variable has

been fixed.

7. Code::Blocks has been removed from the Windows release. The Catalina Geany IDE should be used instead. Code::Blocks was removed from Catalina's Linux release previously, but now that Catalina Geany projects can be provided that support projects of arbitrary complexity (due to the other changes in this release) Code::Blocks is no longer required.

   For those who still want to use Code::Blocks with Catalina, install this release to a different location than the previous release (i.e. do not simply install it over the previous release). If you do this you can continue to use the previous version of Code::Blocks with the Catalina compiler in this release (you need to allow the installation process to update the LCCDIR environment variable, or else do it manually). However, note that any existing Code::Blocks workspaces will still point to the files in the previous release, not the same files in this release. You will need to manually create new workspaces to compile the demo programs included in this release, which are now organized differently.

8. The MORPHEUS and TRIBLADEPROP platforms were Propeller 1 multi-Propeller boards which required many dedicated options and utilities to support. The MORPHEUS had two Propeller 1 chips and the TRIBLADEPROP had three.

   The Catalina support for these boards included:

   Multiple CPU support, including utilities to boot, reset or load programs to RAM or EEPROM from one Propeller to another (e.g. if only one CPU had a serial port to access the external world).

   Proxy driver support, for allowing one CPU to use a display, keyboard, mouse or SD Card connected to another CPU.

   Catalyst support, to allow programs to be loaded and executed on multiple CPUs.

   This release still contains all the build scripts required to build the utilities and demo programs for these platforms, but these are now deprecated and will be removed from a future release. If you have one of these boards, you will need to maintain them yourself in future releases.

9. The Propeller 1 basic and embedded target packages had not been updated for recent releases, and would not compile with the spinnaker Spin/PASM compiler. Also, the *basic* target package has been renamed to *embedded* to better reflect its intended purpose, and also to prevent confusion with the Dumbo Basic program.

10. If the target name was specified using the **-T** command-line option to Catalina, it was not being passed to or used by the Optimizer, which prevented the Optimizer being used for programs built with any but the default target.

11. A bug displaying arrays of chars (or unsigned chars, or int_8) in the BlackBox debugger has been fixed.

12. The flash_payload script now automatically adds the **-o2** flag to payload since this makes FLASH programming on the Propeller 2 more reliable.

13. Dumbo Basic was failing to compile on the Propeller 1 after the last release or two. Now fixed.

14. Many minor documentation updates, refreshes or corrections.
15. Some hidden dependencies on MinGW have been eliminated. This was especially true of some of the Catalyst demo programs. If you install **make** (see below) then you should now be able to build all the demo programs without needing MinGW installed. However, note that you still need MinGW installed to rebuild Catalina itself.
16. 19. Fixed a bug in the NMM threaded kernel, that may have prevented threaded NATIVE programs executing correctly if they were loaded dynamically (e.g when compiled as part of a multi-model program or when loaded from an overlay file).
17. The build scripts and tutorial for the multi-cpu platforms (TribladeProp and Morpheus) have been revised and updated. Also, a bug in the proxy SD Card driver, used in some of the mult-cpu demo programs has been fixed.

Release 4.8

1. Updates to the reference manuals and the command summaries to include the changes made to payload in release 4.7.
2. A bug introduced in release 4.7 when making **NATIVE** the default memory model on the P2 has been fixed.
3. The definition of the **PASM()** function (used to include inline PASM in C code) has been modified to indicate it can return an int rather than a void. This prevents Catalina from overwriting return values (i.e. the value present in r0 when the PASM code finishes execution). The new definition is:

   ```
   extern int PASM(const char *code);
   ```

   This should not make any difference to existing code that does not use the return value of the PASM function. New PASM example program called *test_inline_pasm_3.c* and *test_inline_pasm_4.c* have been added to the examples in the *demos\spinc* folder to illustrate the use of the return value.
4. Updated the **build_all** scripts in the *demos\spinc* folder to only build demo programs appropriate to the propeller type (i.e. 1 or 2).
5. The file *flash_led.obj* was missing from the *demos\spinc* folder.

Release 4.7

6. p2_asm now enables the **-v33** option to p2asm by default. This enables the latest PASM assembly code, and would nearly always be what you want to do. To revert to the previous behaviour, edit the file p2_asm.bat.
7. Dumbo basic updated to version 1.0, which implements much more of the original GWBASIC syntax. See the *README.TXT* file in the *catalyst\dumbo* directory for more details.
8. Added the 8 port serial driver. This driver is only supported on the Propeller 2. It supports up to 8 serial ports on the Propeller 2. See the Catalina Propeller 2 Reference Manual for more details.
9. Fixed a bug in the 2 port serial driver that meant the second port would not receive characters correctly.

10. Fixed a bug in the serial2 and serial4 demo programs which meant they may not have worked if they were compiled with threading enabled.
11. The build_all scripts in all the serial demo directories (i.e. serial2, serial4 & serial8) will now only build the demos for the correct propeller. The serial2 and serial8 libraries are only supported on the Propeller 2, and serial4 is only supported on the Propeller 1.
12. Modified the *catalina_cog.h* include file to fix a name collision with propeller2.h. On the Propeller 2, library functions that are defined in propeller2.h are now not also defined in *catalina_cog.h*
13. Added *smartpins.h* to define the smartpins modes for the Propeller 2.
14. Added an implementation of **_muldiv64()**, to match the SPIN MULDIV64 function.
15. NATIVE mode is now the default memory layout on the Propeller 2. To reinstate the previous default (LMM), specify either -C TINY or -x0 on the catalina command line.
16. The payload loader has two new options:

    -o can be used to override the default propeller version detection. This option requires the propeller version to be explicitly specified i.e. as -o1 or -o2 (propeller 1 or propeller 2).

    -j can be used to completely disable lfsr checking (propeller 1 only).

    This allows payload to correctly use the 100ms serial window when loading programs to a Propeller 2. In particular, it allows programs to be loaded to the FLASH RAM on the P2_EVAL board and executed without requiring the microswitches on the PCB to be altered or the Propeller to be reset.
17. Code::Blocks is no longer supported under Linux, and precompiled binaries are no longer provided. However, the source code modifications and example workspaces for Code::Blocks version 17.12 are still provided for those who want to compile Code::Blocks for themselves from source. Code::Blocks remains supported under Windows, but this may not remain true for future releases. Code::Blocks users under both Windows and Linux are encouraged to migrate to the Catalina Geany IDE, which will remain supported on both Linux and Windows.
18. Under Linux, the srecord utility used is now version 1.64. Under Windows srecord version 1.47 is still used. This utility is used if the -F option to catalina is used. Tests on various common output formats have shown no differences (e.g. Motorola, Intel formats) but exhaustive testing has not been conducted on all formats.

Release 4.6

1. The Catalina Parallelizer is now an integral component of Catalina. To invoke the parallelizer, use the **-Z** option in any Catalina command. Note that this option is positional - it will enable parallelizing all C source files in the command after it appears. You can also use **-z** (i.e. with a lower case z) to stop parallelizing source files. This means the Parallelizer can now easily be used in Code::Blocks or Geany IDEs.

2. Added the *demos\parallelize* subdirectory, with documents and examples that demonstrate the Parallelizer.
3. Added the option of named exclusive code segments to the Parallelizer - all segments with the same name will all use the same lock, which means only one worker can be in ANY of the exclusive zones that specify that name. The name is optional, and _region will be used if no other name is specified.
4. Added more error checking to the Parallelizer, and reformatted the error messages to use the same format as the compiler. Also, the line numbers will now generally refer to the input C source file, not the output C source.

Release 4.5

1. Fixed a bug in the lock processing logic that meant locks may not have worked correctly in multithreaded programs. Affected the Propeller 2 only.
2. Fixed a bug in the "dining philosophers" demo programs that prevented the programs from exhibiting the expected deadlock behavior.
3. Modified various versions of the "dining philosophers" demo programs to allow most options to be configured on the command line.
4. Fixed a typo in the _thread_stop() library function that prevented it from compiling correctly.
5. Added a workaround to a bug in the dynamic memory management library functions (i.e. malloc(), free() etc) that meant programs would not execute correctly in some circumstances. Affected the Propeller 2 only.

Release 4.4

1. The NATIVE code generator for the P2 was not disabling interrupts during CORDIC multiply and divide operations, which might cause a program to fail if an interrupt occurred between CORDIC instructions. This only affected P2 NATIVE programs.
2. Fixed a bug in the multithreading kernel, which may have led to threaded programs locking up. This only affected P2 NATIVE programs.
3. Catalina releases 4.2 and 4.3 were missing the Code::Blocks demo workspaces and projects. Both Code::Blocks and the demos themselves were included, but the workspaces and project files (usually in the *codeblocks* subdirectory of the Catalina installation) were missing.
4. Catalina releases 4.2 and 4.3 included an older version of the Code::Blocks "New Project" wizard, which did not include an option for the FLIP platform. This has been corrected.
5. The Catalina symbol **NO_INTERRUPTS** has been added, which can be defined on the command-line to tell the compiler that the program does not use interrupts – this allows some optimizations to be performed which can improve program performance. This option applies only to P2 NATIVE programs. For example:

   ```
   catalina -p2 program.c -lc -C NATIVE -C NO_INTERRUPTS
   ```

   Note that you cannot use this symbol in P2 NATIVE programs that use threads or interrupts.

6. The Catalina symbol **FAST_SAVE_RESTORE** has been added, which can be defined on the command-line to tell the compiler to use fast block moves when saving and restoring registers. This can improve program performance, but comes at a cost of significantly increased stack usage (since all registers are saved, not just the used registers). This option applies only to P2 NATIVE programs. For example:

```
catalina -p2 program.c -lc -C NATIVE -C FAST_SAVE_RESTORE
```

Release 4.3

1. Implemented the _thread_ticks() function for Native mode on the P2. The version in previous P2 releases was not fully implemented, which meant all multi-threaded programs on the P2 used the default tick count for determining when to context switch. Affected the P2 only.
2. Fixed a bug in the compact version of _sbrk(), which meant that attempting to allocate very large blocks of memory (using malloc) might cause the program to crash. Affected both the P1 and the P2.
3. Fixed a bug which meant that the optimizer could not be used with the VGA HMI option included in Catalina release 4.2. Affected the P2 only.
4. Added the demo\sieve directory, with various versions of the Sieve of Eratosthenes, to demonstrate converting a sequential algorithm to a parallel algorithm.

Release 4.2

1. Added the Catalina Geany IDE. For details see the document "Getting Started with the Catalina Geany IDE". This new IDE was released as an addition to the Catalina 4.1 relase, but will now remain part of all future releases.
2. 6.Added a new HMI plugin for the P2, which supports a VGA monitor with configurable resolution and colour depths, and also a USB keyboard and mouse. It expects to use the P2-ES A/V and Host Serial accessory boards.

   The VGA driver supports 640x480, 800x600 and 1024x768 resolution, and 1 bit (i.e. monochrome), 4 bit, 8 bit or 24 bit colour.

   Updated test programs for the new HMI plugin (test_terminal.c, test_vga.c) are included in the demo folder, and a precompiled version of Catalyst that uses the new HMI plugin is included in the file P2_EVAL_VGA.ZIP in the main catalina folder. Unzip these files to an SD card and insert it into the P2_EVAL board with the A/V (VGA) accessory board on the header with base pin 32, and the Host Serial (USB) accessory board on the header with base pin 40. A keyboard can be plugged into either USB port, but none of the programs use a mouse. Note that the only program that makes use of color is the vi text editor.
3. Increased the size of the P2ASM symbol table to 10000 symbols – otherwise compiling Lua with the new VGA plugin could run out of symbol space.
4. 8.Fixed an error in the Catalina Geany IDE shortcut, which assumed Catalina was installed in the default location. The shortcut and menu item would fail to work if it was not.

5. The build_all.bat script for building Catalyst under Windows was defining the symbol VT100, which would have made some programs – such as vi – not work correctly with the VGA or TV HMI options (it worked correctly with serial HMO options, and now should be manually specified if the PC or TTY HMI options are used).
6. The Catalyst version of vi (xvi) now uses colours (if compiled with a HMI option that supports it).

Release 4.1

1. Added Multi-Memory Model (MMM) support. See the Catalina Reference Manual for details.
2. Fixed a bug in the Makefiles for compiling the **XMM LARGE** libraries. Some library functions were being compiled as **XMM SMALL**, not **XMM LARGE**.
3. Added a new option (**-B**) to the **spinc** utility, for producing blobs from a Catalina C binary file. This option accepts a parameter to specify the object, but for Catalina Propeller 1 programs this is generally the second object, so it is normally specified as **-B2**. The existing command-line options **-c**, **-f**, **-s**, **-l** and **-n** are also supported for blobs, and the -c option would normally be specified to generate a callable function that can be used to execute the blob as a C program. The **-s** option can be used to specify the runtime space (stack and heap) required. If not specified, then 80 bytes (20 longs) is used, which is sufficient for small programs.

   The output is usually redirected to a header file, which is suitable for inclusion in another program. For example:

```
spinc hello_world.binary -B2 > hello_blob.inc
spinc hello_world.binary -B2 -c -s 1024 > hello_blob.inc
```

4. Fixed a bug in catalina that meant the **-M** memory size option (previously used mainly when compiling **EEPROM** programs, but now also useful when generating blobs) was not being specified correctly. This could result in the compilation failing.
5. Modified the Catalina **-M**, **-P** and **-R** options to accept hex values as well as decimals. A hex value must be preceded by **$** or **0x** - e.g. **$ABCD** or **0xFFFF**. Note that these options also accepted modifiers - i.e. **m** or **k** (or **M** or **K**) when using decimal values, but these are not supported for hexadecimal values. Examples of acceptable parameter values are:

```
-M 16k      (or -M16k)
-M 16384    (or -M16384)
-M 0x2000   (or -M0x2000)
-M $2000    (or -M$2000)
-P 1m       (or -P1m)
-P 1048576  (or -P1048576)
-P 0x1000   (or -P0x1000)
-P $1000    (or -P$1000)
-R 8K       (or -R8K)
-R 8192     (or -R8192)
-R 0X3000   (or -R0X3000)
-R $3000    (or -R$3000)
```

This can be useful when generating blobs that must execute at a specific nominated Hub RAM location. Generating a blob is done just by specifying the address of the read-only (code and initialized data segments) to a specific value (i.e. using the **-R** option). The read-write segments will follow immediately, and so the **-P** option does not usually need to be used.For example, to generate a version of hello_world that executes at Hub RAM location 6000 (hex):

```
catalina -R 0x6000 -lci hello_world.c
```

Note that it is the users responsibility to ensure that there is sufficient space for the program code and data, and that this does not overlap with the reserved areas in the upper Hub RAM. The amount of reserved space varies depending on the memory model, plugins and loader options used, so it is recommended that the program itself determine (and check!) this location at run-tine. Examples of how to do this are provided (see the directory *demos**multimodel*).

6. Added **_cogstart_C** (and **_cogstart_C_cog**) to the P1 libraries, which are compatible with the same functions on the P2 (previously the P1 had only **_coginit_C**, but the P2 had both **_coginit_C** and **_cogstart_C**).

   The main difference between **_coginit_C** and **_cogstart_C** is that the latter accepts a parameter, which is passed to the function being started.

   For backward compatibility, **_coginint_C** (and also **_coginit_C_cog**) can still be used if no parameter is required, and **_cogstart_C** (and also **_cogstart_C_cog**) can be used where a parameter is required.

7. Added new functions for starting secondary programs from within a primary program (see the section on Multi-Model support in the Catalina Reference Manual):

   On the Propeller 1 and Propeller 2:

_cogstart_CMM()	start a blob as a CMM program on any available cog
_cogstart_CMM_cog()	start a blob as a CMM program on a specific cog
_cogstart_LMM()	start a blob as an LMM program on any available cog
_cogstart_LMM_cog()	start a blob as an LMM program on a specific cog

   On the Propeller 2 only:

_cogstart_NMM()	start a blob as an NMM program on any available cog
_cogstart_NMM_cog()	start a blob as an NMM program on a specific cog

   However, note that these functions do not generally need to be called directly - when generating a blob that represents a dynamically loadable program, the **spinc** utility can also generate a function to start this program, which knows all the necessary program details.

8. Modified the LMM and CMM dynamic kernels to initialize R2, assuming this value is passed in. This allows **_cogstart_C** (and **_cogstart_C_cog**) to be implemented on the P1 the same way it is implemented on the P2, and also allows **_cogstart_CMM** and **_cogstart_LMM** to accept the address of the shared variable.
9. Allow the **-M** option to be passed to the assembler when compiling CMM or LMM programs (previously, it was only supported for EMM, SMM, XMM etc). The reason this was not previously allowed is that it is not possible for CMM or LMM programs to be larger than 32k. However, when compiling such programs for dynamic loading, it can occur that the program code is to be located in upper Hub RAM, in an area that pushes other things normally included after the program code (such as the kernel) over the 32k limit. So now this option is permitted. But it should only be used for LMM or CMM programs when they are being compiled for dynamic loading via the Multi-Memory Model support (see below). Also, there is no reason to ever use a value other than 64k for such purposes.
10. Added a new symbol for the Parallax FLiP Module. The FLiP Module has no special configuration options, but in case a user chooses to modify the Custom platform, when compiling programs for the FLiP Module the symbol **FLIP** can now be specified on the Catalina command line. For example:

    ```
    catalina hello_world.c -lci -C FLIP
    ```

    Note that the symbol used is **FLIP**, not FLiP! By convention, all Catalina symbols are all upper-case.
11. Fixed a minor bug in the dynamic CMM kernel, which was not registering itself properly on startup.
12. Fixed a problem with **spinc** on Linux that may have led to a segmentation fault when used on some binaries.
13. Removed a limitation in the Compact code generator that prevented program sizes larger than 256k or executing at memory locations above 0x3FFFF.
14. The **-v33** flag to **p2asm** was not being specified in the file *Makefile.inc* in the source\lib directory. This may have prevented some multi-threading functions from compiling correctly for the P2 RevB silicon. Note that this flag will have to be removed again from both this file, and also from the files **p2_asm** (Linux) and **p2_asm.bat** (Windows) in the bin directory to rebuild Catalina for the P2 RevA silicon. Out of the box, Catalina now only supports the Propeller 2 RevB (or later) silicon.
15. Some of the multi-threading demo programs were not calling the function **_thread_set_lock()** before using other thread functions. This worked ok on the Propeller 2 RevA silicon, but will not work on the RevB silicon. Various demo programs have been updated.
16. Modified the various serial libraries to store their lock in the registry instead of in a local variable. This is necessary to support the new Multi-Memory Model, otherwise different kernels executing under different memory models would have used different locks, making the lock pointless. There should be no visible difference in the program behavior.

17. Reduced the number of threads in some of the demo programs when compiled for the Propeller 1 - some combinations of HMI drivers on some platforms occupy more memory than others, which meant the demos (which intentionally push the limits) may have failed to work correctly on some platforms with some HMI options.
18. Fixed a bug in the compact Catalina Optimizer that may have led to some symbols being removed from the source even though they were used – this would have led to 'undefined' symbols being reported when the source was compiled.
19. Added a Code::Blocks workspace to demonstrate building a multi-model program (Windows only) using Code::Blocks.
20. Fixed a potential problem with the Catalina Optimizer, which may have led to relative jumps being used when the target of the jump ended up out of range due to other optimizations.

Release 4.0

1. Updated Code::Blocks to include the **XEPROM** option. Now you can select **XEPROM** in the Catalina Project Wizard (by selecting the menu entry for File->New->Project and then selecting "Catalina Project") and you can also use 2 new tools for loading XEPROM projects (by selecting the Tools menu entry)
2. Added a new named platform to *Catalina_platforms.inc*. This is the **P2_CUSTOM** platform, which by default is identical to the P2_EVAL platform, but which is intended to be customized for new Propeller 2 boards.
3. Because the support for the Propeller 2 is (at least at present!) much simpler than that for the Propeller 1 (e.g. it does not require XMM support to run large C programs, and complex loaders to load them), the Catalina Reference Manual now has two versions - one specifically for the original Propeller 1 (which removes most of the details that are relevant only to the Propeller 2), and another for the Propeller 2 (that removes most of the details that are only relevant to the Propeller 1).
4. Moved the inline PASM demos to the *spinc* folder, and updated the *README.TXT* in that folder. Also added the program *test_inline_pasm.c* as a project in the Code::Blocks "*Spinc Demos*" workspace.
5. Added **tty_txcheck()** functions to the tty and tty256 libraries, to match the **s4_txcheck()** function recently added to the s4 library. These functions all return the number of spaces left in the transmit buffer. The important point is that if they ever return 0, it means a call to the corresponding tx function may block.
6. Fixed a bug that affected Propeller 1 XMM programs that used the cache. Typically, it meant that a program would work with a smaller cache size (e.g 1K, 2K or 4K) but not with a larger cache size (e.g. 2K, 4K or 8K) even though there was enough Hub RAM to accommodate the larger cache.
7. The batch files *p2_asm.bat* and *p2_asm* have been modified to include the -v33 flag to the p2asm assembler. This enables the differences in the instructions set in the latest (and hopefully last) version of the Propeller 2 chip

used in the rev B version of the P2_EVAL board. Note that Catalina itself does not use the different instruction sets, but some plugins or user programs may do so. If it is ever necessary to revert to the previous instruction set (e.g. to run a program on a revA version of the P2_EVAL board), these files must be edited to remove this flag.

8. Fixed a bug where the plugin type of the kernel cog was being overwritten during registry initialization, so the cog would appear to be unused.
9. Added the 2 port serial driver as a plugin, and a library (libserial2) to use it. The 2 port serial driver is only supported on the Propeller 2. The functions in the 2 port serial library are similar to the existing libtty, libtty256 and libserial4 functions. Catalina will load the 2 port serial plugin automatically if you compile your program with the **-lserial2** option. Note that you should not use the 2 port serial driver with the TTY HMI option - they will conflict. This means that when compiling with **-lserial2** you should also either specify **-C NO_HMI**, or use another supported HMI option. For instance:

```
catalina -p2 test_serial2.c -lci -lserial2 -C NO_HMI
```

10. Fixed a problem with the Code::Blocks **Download to Hub Ram and Interact at 23400 baud** tool, which was not setting the directory correctly, and so would sometimes not find the file to download. After installing, you may need to run the *Reset_CodeBlocks* script (found in the Catalina *bin* directory).

Release 3.17.2

1. Updated *Catalina_SD_Plugin.Spin2*, to fix a potential problem when both the clock and the SD card are used intensively - in such cases, the clock may not have had a chance to be updated, leading to the time being out of date. This only affected the Propeller 2.
2. Updated the *test_p2.c* program in the *demos\p2* directory, to fix a potential lockup when testing the locks.

Release 3.17.1

1. Updated *payload.c*, *payload.exe* (Windows binary) & *payload* (Linux binary) to accept any version of the Propeller 2. The previous version only accepted "Prop_Ver A"
2. Update *cogserial.pasm* to the latest version. This caused a name conflict with *BlackCat_DebugCog.Spin2*, so that file has been updated as well.
3. Recompiled the Catalyst binaries in *P2_EVAL.ZIP* to use the latest version of *cogserial.pasm*.

Release 3.17

1. Added a new floating point maths library (**libmc**) which uses the P2 built-in CORDIC solver to allow all the floating point functions to be fitted into a single cog. It is also much faster than the other floating point options. The CORDIC library is is selected by specifying **-lmc** on the command line (i.e. instead of **-lm**, **-lma** or **-lmb**). This option is only available on the Propeller 2.
2. Added a new *propeller2.h* header file, and all the functions specified therein to the standard libraries (*libc*, *libci*, *libcx*, *libcix*). See the file *propeller2.h* in the include directory for more details. This header file (and associated library

functions) will only compile correctly and run on the Propeller 2. A new demo subdirectory (*demos\p2*) has been created with programs that test these functions. The P2 clock demo program *test_p2_clock.c* has also been moved to this subdirectory.

3. On the Propeller 2, Catalina now uses the same locations as Spin to store the clock mode and frequency (i.e. **$14** for clock frequency & **$18** for clock mode). However, note that these are set up AFTER the Propeller program has been loaded, so the option that some loaders (like **loadp2**) have to change these values during the load process should NOT be used with Catalina programs.
4. Added function **s4_txcheck()** to libserial4, which returns the number of bytes available in the tx buffer for the port specified. This can be used to tell whether a call to **s4_tx()** would block (i.e. if **s4_txcheck()** returns 0). The test program in *demos\serial4* has been updated to test this function, but it requires both a TTY and a TV output. Note that the 4 port serial driver is only available on the Propeller 1.
5. Fixed a bug in the 4 port serial driver - unused ports were not being initialized, and so could interfere with the used ports or cause memory corruptions. Also, updated the driver to incorporate some CTS/RTS handling bug fixes by the original author. Note that the 4 port serial driver is only available on the Propeller 1.
6. Added support for executing XMM code from EEPROM on the Propeller 1, which is enabled via the **XEPROM** command-line symbol. The resulting programs can then be loaded using the **EEPROM** option to payload, once the utilities have been built for your platform. The **XEPROM** option is IN ADDITION to any other XMM option you have on your platform, and works on any Propeller with an EEPROM larger than 32Kb.

   XEPROM is only supported for the **SMALL** memory model (i.e. code in EEPROM, data in Hub RAM) and also requires the use of the cache.

   For example, if you have a QuickStart board, after you have used the **build_utilities** batch file to build the EEPROM loader, you can compile, load and execute the *othello.c* program as XMM executed from EEPROM by using the following commands:

   ```
   catalina othello.c -lc -C TTY -C QUICKSTART -C SMALL -C XEPROM
   -C CACHED
   payload EEPROM othello.binary -i
   ```

   For more information, see the XEPROM_README.TXT file in the "target" subdirectory.
7. On the Propeller 2, once initialization is complete and the C code is executing, only the upper half of the LUT is now used (to hold common library code). This means that the lower half of the LUT (i.e. Cog address $200 - $2FF) is available for C applications to use for any purpose if required.

Release 3.16

1. Added Lua scripting capabilities to the **payload** and **blackbox** programs. This is primarily done to assist validation (see next point).

2. Added a new *validation* subdirectory, with scripts and programs that can do automated validation of a Propeller platform. This feature makes heavy of the new Lua scripting feature. See the *README.Validation* file in that folder for more details.
3. Added two new environment variables - **PAYLOAD_PORT** and **BLACKBOX_PORT** - that can be used to specify the port to use for the **payload** program loader and the **blackbox** debugger. This is especially useful when validating.
4. Added the **P2_EVAL** platform to the **build_catalyst** script (this script is primarily intended to be used from Code::Blocks), and also added the ability to specify additional options when building catalyst. Also, building Catalyst now first deletes any existing binaries from the various Catalyst subdirectories - this prevents the possibility of loading old P2 binaries after subsequently recompiling Catalyst for a P1.
5. Removed support for the Homespun spin compiler. The Spinnaker Spin compiler has been the default for the P1 for many releases, and is now the only Spin compiler supported for the P1 (p2asm is used for the P2).
6. The size of the programs that the Pascal Interpreter can load and run has been reduced to 50,000 bytes to enable it to run on more P1 XMM platforms. It is 100,000 bytes on the P2.
7. The library variants (*libc*, *libci*, *libcx*, *libcix*) have changed a little in this release. Now, streams are fully implemented in *libc*, *libcx* and *libcix*, but still only implemented for **stdin**, **stdout** and **stderr** in *libci*. The reason for this is that some functions (such as **vsprintf()**) did not work in *libc* because they relied on streams internally. This limitation saved a few hundred longs, but may have led to unexpected results. Now, only *libci* has this limitation, in order to help reduce program sizes.

   Note that the previous library characteristics can be reinstated by editing the *Makefile* (*Makefile.mgw* in Windows) in the source\lib subdirectory, and editing the lines that look like this:

   ```
   libc: ALWAYSRUN
     $(MAKE) -C stdio clean
     $(MAKE) -C catalina clean
     $(MAKE) -C catalina_io clean
     $(MAKE) -B -C catalina CCFLAGS="$(CCFLAGS) -D__CATALINA_SIMPLE_IO=0 ..."
     $(MAKE) -B -C catalina_io CCFLAGS="$(CCFLAGS) -D__CATALINA_SIMPLE_IO=0 ..."
     $(MAKE) -B -C stdio CCFLAGS="$(CCFLAGS) -D__CATALINA_SIMPLE_IO=0 ..."
     $(MAKE) -C libc all
   ```

   Modify the lines to say **-D__CATALINA_SIMPLE_IO=1** instead of **0** and then rebuild the libraries using the **build_all** script in that directory.
8. Fixed a bug in the **floor()** and **ceil()** functions when using *libmb*. The functions worked ok when using *libm* or *libma*. This problem affected both the Propeller 1 and the Propeller 2.
9. Fixed a problem in various floating point functions (e.g. **sin()**, **cos()**, **log()**, **log10()**, **exp()** etc) when using *libma* or *libmb*. The functions worked ok when using *libm*. This problem affected the Propeller 2 only.
10. Added **go** as a synonym for **continue** in the BlackBox debugger.

11. Allowed a pointer variable to be used as an address in the **read hub** command in the BlackBox debugger - e.g. **read hub my_pointer**. Previously the value of such a pointer had to first be printed using a **print** command, and the result manually entered in the **read hub** command.
12. Fixed some problems when printing or updating variables in the BlackBox debugger:
    - Dereferencing pointers when printing variables did not always work, especially when the pointer was within a structure. Now, BlackBox will not attempt to print nested pointers to structures as structures - it will only print the pointer value.
    - Updating 1-byte and 2-byte variables did not always work (4-byte worked).
    - Character arrays were only being printed if the type of the array element was type **char** (i.e. not type **unsigned char** or type **uint8_t** etc).
    - Only the first field of a local structure or union (i.e. one in the current frame) was being printed correctly.
    - Fixed a problem with the BlackBox debugger P2 NATIVE support - breakpoints were being deleted after their first use.
13. Improved the reliability of reading and writing to SD Cards. Now, Catalina will retry failed reads and writes up to 9 times. This can help prevent SD card corruptions, especially on SD cards that falsely indicate a write is complete but then will not respond correctly for some time afterwards to a subsequent read requests.
14. There is a new *demos\dosfs* subdirectory that contains a test program for the DOSFS file system. This test program has its own version of the DOSFS file system, to allow both the test program and the DOSFS file system to be debugged using the BlackBox debugger. A version of this test program is also included in the validation suite.
15. Fixed a bug in **sbrk()** when programs were compiled in TINY mode on the P2. The wrong version of **sbrk()** was being included in the library.
16. The **Reset_CodeBlocks** script, which copies the codeblocks *default.conf* configuration file to the users directory, was erroneously copying a Windows version of the file even on Linux. The Linux version of the script now correctly copies a Linux version of the file.
17. Fixed a bug in the Catalina Optimizer under Linux, which had not been updated for the new default location (*/opt/catalina*), so it would fail unless the **LCCDIR** environment variable had been explicitly set.
18. Added a **-t** option to BlackBox, to allow a timeout to be specified when attempting to open a comms port. The default timeout was 500ms, but Linux sometimes seems to need a timeout of up to 1000ms to open a port successfully.

Release 3.15.4

1. Fixed a problem with **COMPACT** mode that made programs compile incorrectly in all cases.

2. Fixed a problem with the Catalina Optimizer in **COMPACT** mode that meant optimization level 1 did not work correctly (the other levels worked).
3. Fixed a bug in reporting the program code size, which was including one-off initialization code in the code segment size. This code is correctly included in the overall file size, but the actual compiled code segment size was being misreported.

Release 3.15.3

1. This is the fourth Propeller 2 release. It is partly an internal "clean up" and bug fix release for the previous release (3.15.2), but it also extends the P2 support.
2. Add P2 support for the BlackBox debugger. It now supports **TINY** (LMMM), **COMPACT**, and **NATIVE** modes on the P2. The BlackBox debugger will detect the P2 automatically, so it requires no new command-line options. The debugger requires a serial port connection to the P2, which (by default) is implemented on pins 50 & 52. This allows a PropPlug to be used on the P2 EVAL board, plugged on to the top-right I/O pin header.
3. Fixed an issue with BlackBox on both the P1 and P2 that meant in COMPACT mode, the contents of register r0 could not be displayed by name (i.e. using a **read r0** command - it could be displayed by saying **read cog 0**).
4. Add P2 support for the BlackCat debugger. The BlackCat debugger was originally developed by Bob Anderson, and the functionality is very similar to BlackBox, except that it has a graphical user interface. Note that (unlike BlackBox) BlackCat is provided in binary form only, and that BlackCat is a Windows-only application.
5. There is a new **flash_payload** utility, which programs the P2_EVAL Flash with an auto-executing version of any Catalina program binary. This is based on the Flash_Loader_1.2.spin2 program developed by ozpropdev.

   The parameters accepted by the flash_payload utility are identical to those for the **payload** utility, except that the first parameter MUST be the filename to be loaded. For example:

   ```
   flash_payload startrek -p4 -b230400 -z -i
   ```

   Note that to program the FLASH on the P2_EVAL, you must first remove any SD Card, and also make sure that the FLASH microswitch is turned OFF, or payload will report that it cannot find the P2.

   Once programmed, you must turn the FLASH microswitch ON to boot your program automatically from FLASH (the P2 may also require a re-boot). Turn it off again to boot from SD Card, or to load programs serially.

   Note that FLASH programming sometimes fails. If it does not seem to program correctly, try repeating the procedure.
6. Fixed some issues with the Catalina Optimizer P2 support that led to some erroneous results or less than optimal optimization.

7. Added P2 clock definitions to *catalina_cog.h*, and updated the **_clockinit()**, **_clockmode()** and **_clockfreq()** functions to work correctly for the P2, and a new test program (*test_p2_clock.c*) in the *demos* folder that demonstrates it.
8. Fixed a bug in the HMI plugin that caused some floating point operations to fail in programs compiled in CMM (**COMPACT**) mode.
9. Fixed a bug that prevented the Propeller rebooting on program exit in LMM & CMM modes.
10. Fixed a compiler bug that could lead to incorrect results in some circumstances - e.g. when converting floating point values to integers and vice-versa. This caused problems with programs like the "paranoia" IEEE floating point test. Catalina now passes this test, with the exception of some lack of precision in the **pow()** and **exp()** functions, and some incorrect overflow handling related to "inf" values. These defects should not affect most user programs.

Release 3.15.2

1. This is the third Propeller 2 release. It is partly an internal "clean up" and bug fix release for the previous release (3.15.1), but it also extends the P2 support.
2. Added the ability to have C functions as interrupts. New demo programs have been added in the "demos\interrupts" subdirectory. Interrupts can be used in all the supported P2 memory models - i.e. **COMPACT**, **TINY** and **NATIVE**. There is a new header file (*catalina_interrupts.h)* that defines all the interrupt related functions, and a new C library (*libinterrupts*) must be specified on the command line - see the *demos\interrupts* subdirectory for examples.
3. Fixed a few issues with the Catalina Optimizer, which could lead to errors (e.g. undefined symbols) during compilation, and/or sub-optimal optimization in some cases.
4. Fixed an issue with P1 support, that meant that **COMPACT** programs on the P1 could not be compiled with the Catalina Optimizer. The Homespun assembler support is still broken in this scenario, but Spinnaker (which is the default assembler for the P1) now works correctly.
5. Fixed an issue with **spinpp**, which could have caused programs to fail because C macro substitution was being performed in hex constants like $C1 - so if (for example) you had such a constant in your program and then defined the symbol C1, the code generated would contain $1 for this constant instead of $C1.
6. The semantics of the **_lockset()** function on the P1 have been modified. The difference is that **_lockset()** now returns 1 (i.e. TRUE) on success, whereas it used to return 0 (FALSE) which was in line with the SPIN semantics, but misleading for C programs. On the P2, **_lockset()** has always returned 1 on success. The **ACQUIRE** and **RELEASE** macros in *catalina_cog.h* have been updated accordingly, so programs which used these rather than calling *_lockset()* and *_lockclr()* directly will not be affected.

Release 3.15.1

1. **Code::Blocks** is now fully supported for both the Propeller 1 and 2, and under both Linux and Windows. The version of **Code::Blocks** is now 17.12, but this should not make any difference to existing **Code::blocks** projects or workspaces. Note that on Linux you may need to build **Code::Blocks** from source - the required source code changes are minimal, and ae provided in this release.
2. Linux is now fully supported for both the Propeller 1 and 2.
3. The Catalina Optimizer now supports all the Propeller 2 memory models - i.e. **TINY**, **COMPACT** and **NATIVE**.
4. Payload now tries to load both ".bin" and ".binary" files if no extension is specified.
5. Various minor improvements in Catalyst:
   - better key handling in the **vi** text editor
   - the **cat** command has been vastly speeded up
   - improve the **dir** built-in command
6. Since the Propeller 1 and 2 binaries are incompatible, Catalina no longer includes any precompiled binary files for any platform, although the documentation may still say otherwise.
7. Some sanity testing has been done on the Propeller 1 support, and this release should now correctly compile all Propeller 1 programs (the original 3.15 release had some problems in this area). However, not all Propeller 1 platforms and variants have been tested yet, so it is still recommended that the previous Catalina release (3.13.2) be used for the Propeller 1.
8. Note that you can have multiple versions of Catalina installed - you just need to install each to a different location, and set your **LCCDIR** environment variable according to which one you want to use.

Release 3.15

1. This is the initial Propeller 2 (aka Prop2, P2 or p2) release.

   The support for the original Propeller (aka Propeller 1, Prop1, P1 or p1) is essentially unchanged, and should still work the same as in the previous release - but this has not been extensively tested. This release is intended for those who wish to preview Catalina's P2 support. If you only have a P1 chip, you would be best advised to use the previous release of Catalina (3.13.2).

   To compile C programs for the p2, you just specify the option **-p2** to Catalina (note the lower case 'p' - upper case P means something else). The other Catalina command line options remain essentially unchanged, although not all of them are currently supported on the P2. A summary of the command line options relevant to the P2 is contained in the document "Catalina P2 Command Summary"

   For example, to compile *hello_world.c* for the **P2_EVAL** board, you might use a command such as:

   ```
   catalina -p2 hello_world.c -lc -C P2_EVAL
   ```

As usual with Catalina, an option like **-C P2_EVAL** is used to specify the board support package to use for various things (such as pin definitions and clock speeds) - in this case for the **P2_EVAL** board. See the file *Catalina_platforms.inc* in the *target_p2* directory for details on the supported platforms.

The **-p2** option does the following ...

automatically appends **_p2** to the target directory to use. So in this case the command would use *target_p2* instead of *target* (which is the default target for the P1).

automatically appends **_p2** to any libraries specified. So in this case the command would use the library *lib_p2/libc* instead of *lib/libc* (which is the standard C library for the P1).

predefines the Catalina symbol **P2**, and the C symbol **__CATALINA_P2**, which can be used in various source files to indicate the program is being compiled for the P2.

selects **p2asm** as the assembler to use.

The **catbind** program also accepts the **-p2** option (this option is passed automatically when **catbind** is invoked via the **catalina** command).

All of the supported **build_all** batch files will automatically add the **-p2** and also the **-C NATIVE** command line options if a symbol starting with the letters **P2** (e.g. **P2_EVAL** or **P2D2**) is specified as the first parameter. For example, in the "demos" subdirectory, you can use a command like:

```
build_all P2_EVAL
```

All the **copy_all** batch files will detect if a ".bin" file has been produced (which is the default on the P2) instead of a ".binary" file (which is the default on the P1) and copy the appropriate binaries.

2. There are three memory models currently supported on the P2:

TINY - this model is identical to the **TINY** LMM model on the P1. This is the default model, although on the P2 "tiny" is no longer really an appropriate name since the Hub RAM on the P2 is 512Kb. This memory model can be specified by using the option **-x0** or **-C TINY** on the command line. Note that on the P2 you would usually want to use the **NATIVE** memory model described below.

COMPACT - this model is identical to the **COMPACT** model on the P1. This can be specified by using the option **-x8**, or **-C COMPACT** on the command line. Using this option also specifies that the library is prefixed with "compact_", so if you specify all of **-p2**, **-lc** and **-C COMPACT** the library that is actually used will be *compact_lib_p2/libc*

NATIVE - this model is new for the P2. This is specified by using the option **-x11**, or **-C NATIVE** on the command line. This option is automatically selected in many of the **build_all** batch files. Using this option also specifies that the library is prefixed with "native_", so if you specify all of

-p2, **-lc** and **-C NATIVE**, the library that is actually used will be *native_lib_p2/libc*

3. The default assembler used for the P2 is a slightly modified version of p2asm - so the output of the compilation is now a ".bin" file, instead of a ".binary". There is no support for ".eeprom" output on the P2 yet.

   The only changes to p2asm required for Catalina are two definitions in the file *symsubs.h*:

   ```
   #define MAX_SYMBOLS    8000
   #define MAX_SYMBOL_LEN   65
   ```

4. The Catalina optimizer is supported for all the memory models on the P2.

5. The payload program loader now automatically detects if a Propeller 2 is present, and uses the Propeller 2's built-in serial loader if so. There is therefore no need for an option to specify the p2 (which is just as well, because payload uses the -p option to select the port to use). If you have both P1 and P2 chips connected, you will need to specify the port to use.

   Note that the p2 detection seems to require the payload **-z** option for best results, so a command to download a program to a p2 and then invoke the terminal emulator at 230400 baud would be something like:

   ```
   payload -b 230400 -i -z startrek.bin
   ```

   It is also now possible to use the interactive mode of payload even if no files are loaded - in this case the port must be explicitly specified (and some of the other options are ineffective). For example:

   ```
   payload -i -p4 -b 230400
   ```

   The payload loader will automatically try both ".bin" and ".binary" extensions if none is specifed on the command line.

6. P2 platform-specific definitions are all specified in the one file, (called *Catalina_platforms.inc*) in the *target_p2* directory. This is significantly different from the approach used on the p1, which had so many platforms and plugins supported that it seemed best to use separate files for each one. This may change in future, but for the moment just add any new platforms to *Catalina_platforms.inc* - see the examples in that file.

   The P2 target files are also significantly simpler overall those than for the P1 - because the P1 used a mixture of SPIN and PASM to specify and load various plugins, whereas for the P2 everything is done in PASM. The P1 also required all kinds of tricks to get programs of any size to load and run, which are (as yet!) not required on the P2. This means there is as yet no need for a P2 equivalent of the P1's "basic" and "minimal" targets, although this also may change in future releases.

7. There is no multi-threading support on the P2 yet, so the demos in the *demos\multithread* subdirectory will not yet run on the P2.

8. There is no debugger support on the P2 yet, so the demos in the *demos\debug* subdirectory will not yet run on the P2.

9. There are no plugins other than a simple serial driver, so the demos in the following subdirectories will not yet run on the P2:

```
demos\debug
demos\graphics
demos\vgraphics
demos\spi
demos\serial4
demos\sound
demos\tty
demos\tty256
```

10. The *demos\spinc* directory has not yet been updated for the P2 yet.
11. There is no multi-CPU or proxy driver support on the P2 yet.
12. There is no XMM support on the P2 yet.
13. There are no EEPROM or FLASH loaders on the P2 yet.
14. There is no specific SDCARD loader support on the P2, since this is no longer required. Any .bin program can be loaded from the SD Card. To access the SD card, compile with one of the 'extended' version of the standard C library (e.g. using the option **-lcx** instead of **-lc**).
15. Fixed a problem in Dumbo Basic that caused *startrek.bas* to fail.
16. Note that by default, the "super star trek" (**sst**) program uses CR LF as line terminators. To make this program display correctly in the payload interactive mode, the -q1 option to payload needs to be used. For instance:

```
payload -b460800 -i -z -q1 sst.bin
```

17. Minor change to **xvi** to take into account the faster clock speed of the P2. Also, for the moment the VT100 option is automatically selected when building **xvi** for the P2, since there are no other HMI options available than a simple serial VT100 style terminal that will make this program work correctly (e.g. the one used by payload if the **-i** option is used).
18. Added row and column options to the payload interactive terminal. For instance, to set the terminal size to 80 columns and 50 rows, you can now use a command like:

```
payload -i -g80,50 program.bin
```

    You can also specify the values in the environment variables **PAYLOAD_ROWS** and **PAYLOAD_COLS** if you don't want to specify them on every command. If values are specified on the command line, they override the environment variables. For example:

```
set PAYLOAD_ROWS=50
set PAYLOAD_COLS=80
payload -i program.bin
```

19. Added an option set the payload baud rate using a **PAYLOAD_BAUD** environment variable. For instance, to set the baud rate to 230400 baud for all payload command, you use a command like:

```
set PAYLOAD_BAUD
payload program.bin
```

If a baud rate is also specified on the command line, it overrides the environment variable.

20. Two new utilities have been added:
    - **bindump** - dumps binary files in various useful text formats, which can then be used in other Spin or C programs.
    - **spinpp** - a C-like preprocessor specifically for Spin files.

Release 3.14:

This release was never issued.

Release 3.13.2:

1. Updated Code::Blocks to release 13.12. Updated the Catalina Code::Blocks documentation and quick start guide to match.
2. Improved Code::Blocks compiler support for Catalina - now more compiler options are automatically checked for consistency (e.g. you can only select one HMI option, or only select a caching option for a SMALL or LARGE program).
3. Improved Catalina Project Wizard. Now more options can be selected using the wizard. Also, fixed a problem that did not allow selecting the LARGE memory model using the Wizard. One improvement is that it is now possible to specify the output format (.binary or .eeprom). This means Code::Blocks can now correctly tell whether a target needs rebuilding or not.
4. Made it possible to create a single default target using the Wizard. This simplifies things, and is now the recommended way to create projects. The previous method (creating a separate debug and release target) is still supported. The release target (if created) is also now optimized by default.
5. Improved the build scripts to depend only on the LCCDIR environment variable, which allows Catalina to be rebuilt with a single command with no editing of scripts etc required.
6. Added more Code::Blocks example workspaces and projects (using existing C programs from the demos folder) - these are in the workspaces

   sound_demos : sound_spacewar, sound_demo

   more_demos : globbing, sumeria, chimaera, test_spi, test_tty
7. Fixed some problems with some of the Code::Blocks projects, such as the pre- and post- build command in the *spinc_demo* projects.

Release 3.13:

1. Fixed a problem with the graphics library that caused it to not compile if **spinnaker** (openspin) was used. It compiled correctly if **Homespun** was used.
2. Modified the CMM kernel to include relative jumps, and various other minor performance improvements. These can both improve the speed and reduce the size of CMM programs significantly.

3. Add a new Optimizer level (level 5) which optimizes loads. This can improve the speed and reduce the size of both CMM and LMM programs significantly.
4. Improved the performance of the **PSHM** primitive - this can improve the speed of both CMM and LMM programs that make intensive use of procedure calls significantly.
5. The **ALTERNATE** LMM kernel is now deprecated, since the improvements in the standard LMM kernel have made it redundant. It will be removed from future releases.
6. The combined result of the the CMM and LMM changes can reduce program size and improve program speed as follows:

   CMM: up to 20% size reduction, and up to 15% speed improvement

   LMM: up to 5% size reduction, and up to 20% speed improvement.

   Not all programs will achieve these reductions, but all non-trivial programs will achieve SOME improvement.

In addition, the following issues were fixed in a **3.12 Errata** release, and are also fixed in this release:

1. Modified the *sumeria.c* and *chimaera.c* demo programs to seed the random number generator on each execution.
2. Fixed a bug in the CMM kernel in the **PSHB** primitive. However, this primitive is not currently used by Catalina, so this bug would not have affected any existing Catalina programs.
3. Included a new demos for the SD card file system - a "globbing" demo that allows pattern matching of file names - see *demos\globbing* for more details.
4. Fixed some script issues in the **build_utilities** and **Set_Linux_Permissions** that only affected Linux or OSX platforms.
5. Fixed some compilation issues in the source\catalina directory that were preventing Catalina compiling with the clang compiler under OSX.
6. Added the ability to specify the port by name to the **blackbox** program (via the **-p** parameter). This is required only under OSX, where ports must always be specified by name.
7. Fixed a problem with the CMM thread libraries that prevented programs from compiling when the spinnaker Spin compiler was used (programs compiled ok using the Homespun compiler). Only affected **COMPACT** mode programs.
8. Fixed a problem with the CMM optimizer that may have resulted in programs not executing correctly if the Catalina Optimizer was used (programs would execute correctly without the optimizer).

Release 3.12:

1. At the request of the maintainers of the “openspin” open source Spin compiler, Catalina's version has been renamed (from **openspin** to **spinnaker**). Catalina's version will always be compatible with Catalina's other

components, and is currently 100% compatible with **openspin** (the necessary Catalina extensions to the preprocessor are enabled only when the **-a** command-line option is specified) but this may not always remain so in future as **openspin** continues to develop, so the renaming has been done now to avoid confusing users who may also use **openspin**.

2. The Catalina cache operation has been speeded up significantly (in most cases the program execution speed has more than doubled) by re-querying the XMM RAM only when the Kernel knows the page currently loaded in Hub RAM changes - otherwise use the Hub RAM copy. The previous behavior was intended to allow for the possibility of more than one cog executing code from XMM RAM, but this makes the executing speed far to slow to be useful.
3. Catalina now differentiates between base platforms and XMM add-on boards. In previous versions of Catalina, only one platform symbol could be specified on the command line, and this specified both the base platform and the XMM API required to use the XMM RAM on that platform. With the advent of many new "add-on" XMM boards (such as the Propeller Memory Card) that can be added to multiple base platforms, Catalina now allows both a base platform and an XMM add-on board to be specified. The XMM add-on board API will take precedence over any native XMM API. For instance, if you have a C3, but want to use the PMC as XMM RAM instead of the built-in C3 XMM RAM, you can now do so. For example:

```
catalina hello_world.c -lci -C LARGE -C C3          <-- use C3 XMM
RAM

catalina hello_world.c -lci -C LARGE -C C3 -C PMC   <-- use PMC
XMM RAM
```

The Hydra Xtreme (HX512) is also now defined as an add-on board, but for historical reasons this does not need to be specified separately when the base platform is either the Hydra or the Hybrid (although there is no problem with doing so).

To support this new flexibility, there are now several types of board support files in the Catalina target directory, depending on the type of board:

For a base platform (XXX) that has no XMM RAM, there are three files:

```
     XXX_DEF.inc            <-- specifies base pin and clock
definitions
    XXX_CFG.inc          <-- specifies #defines for configuring
plugins
   XXX_HMI.inc       <-- specifies HMI options supported
```

For an XMM RAM add-on board (YYY), there are three different files:

```
   YYY_XMM.inc       <-- specifies XMM API functions
```

```
      YYY_XMM_DEF.inc      <-- specifies XMM pin and memory
definitions
     YYY_XMM_CFG.inc     <-- specifies #defines for configuring
plugins
```

For a base platform (ZZZ) that also has built-in XMM RAM, there will be six files:

```
     ZZZ_DEF.inc            <-- specifies base pin and clock
definitions
    ZZZ_CFG.inc          <-- specifies #defines for configuring
plugins
   ZZZ_HMI.inc       <-- specifies HMI options supported
   ZZZ_XMM.inc       <-- specifies XMM API functions
      ZZZ_XMM_DEF.inc      <-- specifies XMM pin and memory
definitions
     ZZZ_XMM_CFG.inc     <-- specifies #defines for configuring
plugins
```

This mechanism, although slightly more complex than previously, allows XMM add-on boards to be "mixed and matched" with various base platforms - although in general it will still be necessary to adjust the pin definitions when moving an add-on board from platform to platform.

Also, some XMM add-on boards include SD card sockets which may be used in place of existing SD Card sockets on the base platform (in some cases this is NECESSARY, since the pins used on the base platform are used for other purposes on the XMM add-on board). In such cases, the pin definitions on the base platform may need to be edited as well, although this can easily be done in a manner that allows for both the base platform SD socket and the possible XMM add-on board SD sockets. For example, here is the code in the *QuickStart_DEF.inc* file, which knows the pins used for the SD Card sockets for various common add-on boards:

```
      #ifdef PMC
        SD_DO_PIN   = 1                      ' Propeller Memory Card
(override)
        SD_CLK_PIN  = 7                      ' Propeller Memory Card
(override)
        SD_DI_PIN   = 0                      ' Propeller Memory Card
(override)
        SD_CS_PIN   = 4                      ' Propeller Memory Card
(override)
      #elseifdef RP2
      SD_DO_PIN  = 12                ' RamPage2 (override)
      SD_CLK_PIN = 13                ' RamPage2 (override)
      SD_DI_PIN  = 14                ' RamPage2 (override)
      SD_CS_PIN  = 15                ' RamPage2 (override)
      #else
      SD_DO_PIN  = 0                 ' Human Interface Board
      SD_CLK_PIN = 1                 ' Human Interface Board
      SD_DI_PIN  = 2                 ' Human Interface Board
      SD_CS_PIN  = 3                 ' Human Interface Board
      #endif
```

More could be added if needed.

4. Support has been added for the RamBlade3. This is enabled via the **RAMBLADE3** symbol. See the file *RamBlade3_README.TXT* in the Catalina target directory for more details.
5. Support has been added for the Propeller Memory Card. This is enabled via the **PMC** symbol. See the file *PMC_README.TXT* in the Catalina target directory for more details.
6. Explicit support has been added for the QuickStart. This is enabled via the **QUICKSTART** symbol. See the file *QuickStart_README.TXT* in the Catalina target directory for more details. Note that the QuickStart was already supported in previous versions, but via the CUSTOM symbol. Now, it has been allocated a symbol of its own.
7. The pin definitions for the **SUPERQUAD** and **RAMPAGE** have been modified to suit the Propeller Platform (**PP**) since it is more likely these boards will be used with that platform than the previous default (**C3**).
8. Fixed a problem with the SD Plugin introduced with Catalina 3.11, which was not correctly clocking the SD card after raising the Chip Select. This could lead to the SD card plugin not working with some XMM memory add-on boards.
9. Fixed a problem with the Catalina File System call **_close_unmanaged()**, which was not correctly clearing out the file handle. This meant that after a small number of file handles had been used (8 in total), no more files could be opened since it looked like all the available file handles were still in use.
10. Fixed a problem with the special register names (e.g. **DIRA**, **OUTA**, etc) which meant that if constant values were assigned to them, they could be interpreted as cog addresses rather than constants in some circumstances. For instance, **OUTA = 0** might assign the value contained in cog address 0 to **OUTA** rather than the constant value 0. This did not affect inline PASM, or the library functions (e.g. **_dira(), _outa()** etc).
11. Fixed a problem under Windows with catdbgfilegen not being compiled as a "static" binary, which meant it would throw an error about a missing DLL when a program was compiled with the **-g** flag unless MinGW was included in the PATH environment variable.
12. Fixed a problem with the "tiny" library, which was including a copy of the stdio function **puts**. This conflicted with the version of puts in the standard C library if **puts** was used in a program.
13. Added a new tutorial document called "Selecting Catalina Options" and two demo programs, written by Mike Arnautov (http://www.mipmip.org):

    sumeria.c

    chimaera.c

    The document describes how to compile these programs using various different Catalina memory models and load options.
14. The Propeller Memory Card (**PMC**) is also compatible with the **DEMO** board, so *Demo_DEF.inc* has been upgraded to know about the SD Card on the

PMC. To use the **PMC** with the Demo board, just add **both** symbols to the command line. For example:

```
catalina program.c -lcx -C DEMO -C PMC
```

15. Fixed a bug in the **TRIBLADEPROP** XMM API, which resulted in **XMM_IncAddr** being undefined when programs were compiled for **CPU_2**.

Release 3.11:

1. By default, Catalina now uses the **openspin** spin/pasm compiler to assemble the PASM code generated by the compiler. Full sources for this compiler are included, and it is MIT licensed. The openspin compiler has been modified to support some Homespun features needed by Catalina. The openspin compiler has been tested with all Catalina sources, but Homespun is still provided in case any problems are discovered - if there is any need to use homespun, simply use the -a command line option to specify which assembler to use. For example:

```
catalina hello_world.c -lc       <-- assemble with openspin
catalina hello_world.c -lc -as   <-- assemble with openspin
catalina hello_world.c -lc -ah   <-- assemble with homespun
```

   The assembler in use can be detected (if required) in both C and in Spin. This allows code to be written that can cope with differences in the two compilers (Catalina uses this mechanism itself in the target file *Catalina_Common.spin* to workaround a Homespun bug).

   In C:

   The symbol **__CATALINA_HOMESPUN__** or **__CATALINA_OPENSPIN__** will be defined according to the assembler used.

   In Spin:

   The symbol **HOMESPUN__** or **OPENSPIN__** will be defined according to the assembler used.

2. There is a new SD card plugin, which supports SDHC cards (i.e. SD cards over 2Gb in size). This new plugin is used by default. To revert to using the previous plugin, simply define the Catalina symbol **OLD_SD** using the -C command line option. For example:

```
catalina test_catalina_fs.c -lcx            <-- use new plugin
catalina test_catalina_fs.c -lcx -C OLD_SD  <-- use old plugin
```

   On some platforms, the new SD card plugin is much faster than the old one.

3. Fixed a bug in the *test_registry.c* demo program, which was using old names for plugins and would not compile.

4. Added the ability to include pasm instructions 'inline' with C code. This uses a new PASM function, defined in *propeller.h* as:

```
extern void PASM(const char *code);
```

   This is not a real function. Instead, the compiler recognizes this as a request to insert one or more PASM instructions in the assembly output of the compilation. For instance:

```
void main() {
   PASM ("mov R0, #1");
```

```
    PASM ("mov outa, R1");
}
```

See the example program *test_inline_pasm.c* in the demo folder for more details.

5. Updated the *minimal* target to use an *Extras.spin* file, the same as all the other targets. No functional change.
6. Now that **openspin** support is available, support for the **bstc** Spin/PASM compiler has been removed from both the catalina and **catbind** commands.
7. the **catbind** utility now uses -**ah** and **-as** command line flags to specify the SPIN/PASM compiler to be used. This makes it compatible with the main catalina command.
8. Payload now has additional options for managing CR/LF translations. These are specified using additional values to the -q command line option:

   The existing -q values ...

   1 = ignore CR

   2 = ignore LF

   ... can now be combined with ...

   4 = translate CR to LF

   8 = translate LF to CR

   So (for example) ...

   -q6 means ignore LF, but translate CR to LF

   -q9 means ignore CR, but translate LF to CR

   Note that combining 3 (i.e. ignore both CR and LF) with either 4 or 8 will have no effect. For an example program that can be used to understand the effects of the various -q options, see the file *test_payload.c* in the demos directory.

   A common situation where **-q** is required is when C programs that use windows-style line termination are being run in a terminal emulator (such as payload provides via the -i command line option) which expects Unix-style line termination. In this case, use **-q1**.
9. The library function *_coginit_Spin* used a *wait* function but did not define it, so it would fail to compile unless the user provided this function themselves. This has now been fixed so that it uses *waitcnt* directly.
10. Fixed a bug in the FAT file system code (DOSFS) that may have led to corrupt directory entries.
11. Fixed a bug in libserial4 that meant the wrong offsets were being used when multiple serial ports were in use. This may have led to the serial port code apparently working, but corrupting memory used for other purposes.
12. Fixed a bug with the system function *sigjmp* and *siglongjmp* with the Catalina optimizer - since they were both implemented by the same code, if one was optimized away because it was not used, both were being removed even though one was still required.

13. Added a workaround for a Homespun bug that removed what it thought were redundant identical instances of *Catalina_Common.spin*, which were not in fact identical because one included additional layer 2 or 3 functions. Now, when Homespun is used as the assembler, all instances of this object are forced to be the same, even though this incurs a few hundred bytes of overhead.
14. Added support for the RamPage 2 board, via the command line symbol RP2. See the file *RP2_README.txt* for more details.
15. Added support for using TTY on the Triblade Prop CPU #2 and CPU #3 (it was already supported on CPU #1).
16. The **ramtest** utility now also supports the TTY option.
17. The XMM_LOADER symbol was not being defined by default in *DracBlade.CFG* which meant Catalyst was not being compiled with the XMM Loader functionality enabled (unless this symbol was defined on the command line), meaning XMM programs could not be loaded using Catalyst (they could still be loaded serially).
18. The default HMI colors (used for VGA and TV displays) were being set to TV colors when no HMI option was specified on the command line, even if the default HMI option was VGA (e.g. as it is on the DRACBLADE). Now they are set in the various HMI screen drivers instead of in the *Catalina_Common.spin* file.
19. Catalyst is now a C program. The Catalyst utilities are also C programs. In some cases, the utilities now have additional options. The file *Catalina_FATEngine.spin* file has now been removed, since it was only used by the Spin version of Catalyst, and it was becoming too difficult to maintain two completely different file systems. Spin file systems (like FATEngine or FSRW) can of course be used by Spin programs loaded by Catalyst, but Catalina and Catalyst both now only use DOSFS.
20. Catalyst now uses Linux style line termination by default. To restore the Windows style line termination, edit *catalyst.h* in the *catalyst\core* subdirectory and recompile Catalyst.
21. Added **DFS_UnlinkDir** to delete directories - the directory to be deleted must already be empty, since any clusters occupied by any files in the directory are not reclaimed (potentially leading to lost clusters).
22. The **t_geometry()** function in the LoRes TV driver was returning the values for rows and columns swapped around.
23. Fixed a minor bug in the **_sbrk()** function that wasted some available dynamic memory space (only affected TINY and COMPACT modes). Also, fixed a bug in the *test_sbrk.c* test program - the program sometimes failed to terminate due to an uninitialized pointer variable.
24. Fixed a bug in Catalyst that meant SMM files were not loaded correctly on platforms where the Cache and Flash RAM was used.
25. Both catalina and catbind are now quieter than they used to be - the default now is for both to output minimal information messages. The option **-v** (verbose) means to output *some* information (approximately what used to

be output by default) and **-v -v** (very verbose) means output *much more* information (approximately what used to be output for verbose mode).

Release 3.10:

1. Fixed a bug in the CMM Code Optimizer that could lead to optimized programs that behaved differently to non-optimized programs.
2. Added a new **EEPROM** data loader that can be used as the first file of a **payload** command. The EEPROM loader can be used to program files of up to 32Kb into EEPROM (similar to the payload -e command) on any Propeller,but it can also be used to load programs larger than 32Kb into EEPROM on Propellers that have 64Kb EEPROM. This is very useful when the EEPROM loader option is needed to load an LMM program into Hub RAM because the program needs more code space than can be accommodated with the default loader. It can also be used to load an XMM program from EEPROM. To build the EEPROM loader, use the **build_utilities** batch file, and follow the prompts. Then specify EEPROM in a 'payload command'. E.g:

```
build_utilities
catalina othello.c -lci -C EEPROM -C HYDRA
payload EEPROM othello
```

or

```
build_utilities
catlaina othello.c -lci -C LARGE -C EEPROM -C C3
payload EEPROM othello
```

3. Updated the **build_utilities** batch file. This batch file can be called from Code::Blocks or from the command line. It now builds utilities that are more intuitively named:

   EEPROM - load a program into EEPROM of 32k, or 64k or larger. If it is larger than 32kb, the program must have been compiled with the **-C EEPROM** compiler switch, which includes the EEPROM loader in the compiled program.

   SRAM - load a program into SRAM. The program must have been compiled with **-C SMALL** or **-C LARGE**.

   FLASH - load a program into FLASH. The program must have been compiled with **-C FLASH**, plus a cache size (e.g. **-C CACHED_1K**).

   MOUSE - load a program into SRAM via the mouse port - used for HYDRA and HYBRID, where the normal serial port cannot be used when the HYDRA XTREME XMM card is installed. The program must have been compiled with **-C SMALL** or **-C LARGE**.

   XMM - while deprecated, this is retained for backward compatibility. Loads a program into the default XMM RAM configuration, which may be **SRAM**, **MOUSE** or **FLASH** depending on the platform.

   All these utilities are used the same way, as the first file to be loaded via the

   payload serial loader. For example:

```
catalina program.c -C EEPROM
payload EEPROM program
```

or

```
catalina program,c -C LARGE -C FLASH -C CACHED_1K
```

```
payload FLASH program
```

Note that the *build_all* batch file in the utilities folder remains, but it does not use the new names (SRAM, FLASH etc) - it simply builds the default XMM loader as **XMM**. However, the use of this batch file is now deprecated since the *build_utilities* can build all the utilities that this batch file used to build, and is much easier to use. A message to this effect is now displayed if the old 'build_all' batch file is used (but it still works).

4. Added the **SOUND_PIN** to the standard platform definition file for each platform (e.g. *CUSTOM_DEF.inc*). The sound plugin and support library are now compiled for each memory mode as part of the standard set of libraries, and can be enabled by simply including **-lsound** on the command line. For example:

```
catalina sound_demo.c -lc -lsound -C HYDRA
```

Demo programs are included in the 'demos\sound' folder. For more information on the sound library, see *catalina_sound.h* in the 'include' folder.

5. The sound demos (including *spacewar.c*) now run on both the C3 and the HYDRA, and may also work on other platforms. See the *README.TXT* file in the *demos\sound* directory for more details. The version of *spacewar.c* in this directory and the version in in the *demos\vgraphics* directory are identical, but in the *demos\sound* directory the *build_all* batch file adds *-lsound* to the command line, which enables the sound plugin.

6. Generalized the Gamepad plugin to suit any platform, and added the Gamepad Pins (**NES_CL_PIN**, **NES_CS_PIN**, **NES_D1_PIN**, **NES_D2_PIN**) to the standard definitions file for each platform. The Gamepad plugin is so simple to use that it needs little in the way of support functions, so instead of having a separate library, the Gamepad support functions are included in all the standard libraries. To use this plugin, just add the option **-C GAMEPAD** to the Catalina compile command. For more details, see the *test_gamepad.c* demo program, which can be compiled with a command like:

```
catalina test_gamepad.c -lci -C GAMEPAD -C HYDRA
```

7. Added support for the Propeller Platform USB, and also for the 'El Jugador' expansion board. The symbol **PP** enables support for this board. For example:

```
catalina test_gamepad.c -lci -C GAMEPAD -C PP
```

8. Fixed a bug in building the library from source, where the build_index batch files specified */bin/sh* instead of */bin/bash*.

9. Macros to simplify the use of locks - i.e. **ACQUIRE(lock)** and **RELEASE(lock)** have been added to *catalina_cog.h* rather than being redefined within each support library that used them.

10. Added a new SPI plugin, that can be used to communicate with EEPROMs on an I2C bus, or SD Cards on an SPI bus. The plugin is enabled by linking with the support library - e.g. add **-lspi** to the command line. For more information see the *catalina_spi.h* file in the 'include' directory. The plugin is based on an original Spin driver by Mike Green. A demo program for the SPI plugin is available in the directory *demos\spi*. See the **README.TXT** file in that directory for more details.

11. Added a new function to the standard library - **_plugin_name(type)**, which accepts a plugin type (e.g. extracted from the registry) and returns a pointer to a string containing a readable name for the plugin type. This function is now used by several of the demo programs (which previously all had to have their own copy).
12. **BlackCat**, the graphical source-level Debugger for Catalina C programs originally developed by Bob Anderson, has been updated to include support for the new CMM memory mode, and a binary version is included with Catalina (the source is not included - BlackCat is not open source). Note that BlackCat runs only under Windows - however, Linux users can still use the BlackBox debugger, which is a command-line debugger that provides very similar debugging capabilities. There is a new document (**Getting Started with BlackCat**) that gives a brief introduction into the BlackCat debugger, and Code::Blocks has additional entries on the Tools menu to use the BlackCat debugger as well as the BlackBox debugger.

Release 3.9:

1. Fixed a bug in the CMM Code generator that generated incorrect code for division of a signed integer by small powers of 2 (e.g. 1,2,4,8,16).
2. Added *libtty256*, which is a variant of the existing *libtty* but which uses Ray Rodrick's customized version of FullDuplexSerial, with the buffer size increased to 256 bytes. Demo programs are provided in *demos\tty256*
3. In the Linux distributions, the *vgraphics* library was not being built due to a bug in the makefile.

Release 3.8:

1. Added full support for CMM (Compact Memory Model). CMM now fully supports multi-threaded and multi-cog programs, and all the program loaders can now load and execute CMM programs.

   The Compact memory model is enabled by adding the command line option **-C COMPACT**. The CMM memory mode generates code that is under half the size of the equivalent LMM program.

   For example:

   ```
   catalina test_suite.c -lci                => code size = 6812 bytes
   catalina test_suite.c -lci -C COMPACT     => code size = 3196 bytes
   ```

   The Catalina Optimizer also works with CMM mode, although the savings are not currently as dramatic as they are for LMM mode:

   ```
   catalina test_suite.c -lci -C COMPACT -O3   => code size = 3136 bytes
   ```

   Both the Payload loader and the Catalina BlackBox debugger work with CMM mode:

   ```
   catalina test_suite.c -lci -C COMPACT -g3
   payload test_suite
   blackbox test_suite
   ```

   The only known limitations of CMM mode is that programs that use FCACHE cannot use the debugger. However, in this release the only programs that use FCACHE are those that use the graphics library or the threaded kernel.

2. Added the vgraphics library (The "v" is for "Virtual", but it could also stand for "Vector" or "VGA"). The vgraphics library is highly compatible with the existing graphics library, but supports VGA displays instead of TV displays. To compile programs to use the vgraphics library, just add **-lvgraphics** to the command line. For example:

   ```
   catalina graphics_demo.c -lvgraphics
   ```

   The following command line symbols can be defined for programs that use vgraphics, to configure the graphics resolution, color depth and other options:

   VGA_640 - resolution is 640 x 480. This is the default.

   VGA_800 - resolution is 800 x 600.

   VGA_1024 - resolution is 1024 x 768.

   VGA_1152 - resolution is 1152 x 864.

   VGA_2_COLOR - color depth is 1 bit (2 colors).

   VGA_4_COLOR - color depth is 2 bits (4 colors).

   DOUBLE_BUFFER - enable double buffering (smoother graphics).

   Example programs are provided in the *demos\vgraphics* folder.

3. Fixed a bug in the debugger when debugging CMM mode programs – breakpoints set using function names were sometimes set at the wrong address.
4. Added the ability to display Hub RAM above 32kb - this is the ROM space on the Propeller v1.
5. Removed the files *emm_progbeg.s* and *smm_progbeg.s* - emm and smm are really load options, not separate memory modes. The memory modes are lmm, cmm and xmm. Similarly for *emm_progend.s* and *smm_progend.s*
6. Added keyboard support to the graphics library. Now both the graphics and vgraphics libraries have full keyboard and mouse support, and there is no need to use the HMI keyboard and mouse drivers. In fact, when using the graphics or vgraphics libraries, it is generally recommended that the NO_HMI option be used - this prevents the HMI keyboard and mouse drivers from loading, which may conflict with the drivers provided by the graphics and vgraphics plugins. However, it is possible to leave the PC or TTY HMI options enabled, since they use only the serial port. To disable individual graphics or vgraphics keyboard or mouse drivers, you use the same options used with the HMI plugins - i.e. **NO_KEYBOARD** or **NO_MOUSE**.
7. Blackbox now correctly removes the extension if a .binary or .eeprom file is specified when debugging.
8. Some minor changes to Code::Blocks to add support for the vector graphics plugin. The Project Wizard now allows the NO_HMI option to be selected, which is necessary for graphics programs. The **NO_KEYBOARD** and **NO_MOUSE** options (which are common to both the Graphics and HMI plugins) have now been moved to a new "HMI and Graphics Options" category in the build options.

Release 3.7:

NOTE: Catalina 3.7 was a BETA release that introduced preliminary support for the Compact Memory model – there was no non-beta release of 3.7. Full support for CMM support was released as Catalina 3.8.

1. Added support for CMM (Compact Memory Model). The CMM model is enabled by using the command line option **-C COMPACT**. The CMM memory mode generates code that is typically around half the size of the equivalent LMM program.

   For example:

   ```
   catalina test_suite.c -lci                 => code size = 6812 bytes
   catalina test_suite.c -lci -C COMPACT      => code size = 3408 bytes
   ```

   The Catalina Optimizer also works with CMM mode, although the savings are not currently as dramatic as they are for LMM mode:

   ```
   catalina test_suite.c -lci -C COMPACT -O3   => code size = 3348 bytes
   ```

   Both the Payload loader and the Catalina BlackBox debugger work with CMM mode:

   ```
   catalina test_suite.c -lci -C COMPACT -g3
   payload test_suite
   blackbox test_suite
   ```

2. Known limitations of CMM mode in this BETA release:

   a) Programs that use FCACHE cannot use the debugger.

   b) The threads library is not yet supported by the CMM kernel.

   c) CMM mode programs can only currently be loaded serially.

   d) XMM RAM is not yet supported by the CMM kernel.

Release 3.6:

1. Added a new option (-q) to control the interpretation of line termination characters sent by the Propeller program to the Payload interactive terminal emulator:

   -**q1** : ignore CR

   -**q2** : ignore LF

   -**q3** : ignore both CR and LF

   The option **-q1** is useful when running programs that use DOS line termination (i.e. CR LF) with the Payload terminal emulator, since the CR effectively causes these lines to be erased as soon as they have been displayed.

2. Fixed the Ram_Test utility (it wouldn't compile in Catalina 3.5). Also, there is now only one source file for the RAM Test program (although the build_ram_test batch file still builds PC, TV and VGA versions).

3. A new image format has been implemented, which typically saves around 300 bytes of HUB Ram for all memory modes (LMM or XMM). Also, the binary format for Flash programs are much smaller - around the same size as the non-flash versions.

4. In conjunction with the new binary formats, Payload has been modified to load programs faster. In some cases, payload can even load TINY programs faster

than the Parallax loader. All XMM programs load much faster, especially those that use Flash RAM. In most cases, loading an XMM version of a program is now as fast - or faster - than loading the equivalent LMM version.

5. Catalina now compiles files "in place" - i.e. in the users current working directory, and not in Catalina's target directory. This now makes it possible to have multiple users using Catalina at once (note that they must be compiling in different directories) and also makes Catalina more compatible with Windows 7 and Vista, which do not like programs that write to the "Program Files" directories. The target directories that are used when compiling C programs can now be made "read only", and normal users no longer need to have Administrative rights to use Catalina.
6. Catalina now installs the bin directory and all target directories as read-only for normal users and read-write for administrative users. This means that users who need to modify target files, or build the utilities and have the results copied to the bin directory, will need to have administrator rights. But note that users who just want to use Catalina to compile C programs do not need administrator rights.
7. Users without administrative rights can now compile the catalyst and the utilities by first copying the respective directories to their home folder. The **build_utilities** and **build_catalyst** batch files provided now check for the existence of these folders in the user's home directory and compile the catalyst and utility programs in those folders if found. Note that when compiling the utilities folder, the last thing the batch file does is attempt to copy the resulting binaries to the Catalina bin folder - this copy will fail if the user does not have administrative rights. However, the binary files can instead be copied to the same folder in which the user is compiling their C programs, since Payload checks this location first for any loader files it needs (e.g. **XMM.binary**).
8. A new batch file, called **Reset_Codeblocks** is provided. Executing this batch file will overwrite the current Code::Blocks configuration file with a new copy. This is especially useful when using Code::Blocks as a user other than the one who installed Catalina, as the Tools menu will initially be empty in that case. To rectify this, simply execute this batch file. It can be re-executed at any time to reset the Code::Blocks configuration back to the original state.
9. Fixes a bug in Payload - it now always uses the first port for interactive mode, not the subsequent port (when two different ports are used). This affected the Hydra and Hybrid platforms when using the Mouse Port Loader.
10. Fixed a bug in Payload where the new interactive mode was not being started when the verbose or diagnose command-line option was also specified.
11. There is now only one XMM kernel debugger overlay for all XMM platforms. Previously, a different overlay was required for each platform, which complicated the development of new XMM platforms. Now, debugger support is automatic for any new platforms that provide the standard set of XMM API functions.
12. Smaller and faster division functions have been included. This should not make any functional difference to any existing programs.

13. Fixed a bug in the use of Catalina environment variables. In particular, the **CATALINA_LIBRARY** environment variable should not now specify the whole path, since the last component of the library path (*lib* or *large_lib*) must be determined by Catalina based on the memory model specified for each compilation.
14. Fixed a configuration problem where Catalyst would not build for non-XMM platforms. Catalyst works on any platform, but should have disabled XMM support when compiled for platforms that do not have XMM RAM. This is now managed by defining the **XMM_LOADER** symbol in the *platform_CFG.inc* file for each platform.
15. Fixed a problem with the definition of **clockmode()** in <*catalina_icc.h*>.
16. Added support for direct access to the special cog registers, in addition to the existing access (.e. via function calls like **_ina()** etc). The new include file <*propeller.h*> defines the variables **PAR**, **CNT**, **INA**, **DIRA** etc as **external volatile** variables, and they can then be used like any C variable. For instance:

```
x = _ina()       is now equivalent to  x = INA
x = _get_dira()  is now equivalent to  x = DIRA
_dira(m,d)       is now equivalent to  DIRA = ((DIRA & m) | d)
_dira(0,d)       is now equivalent to  DIRA |= d
_dira(m,0)       is now equivalent to  DIRA &= ~m
```

    The <*propeller.h*> file also contains other useful definitions to assist Spin programmers who want to try C.
17. The **DISABLE_REBOOT** symbol has been renamed **NO_REBOOT** to be more consistent with other such symbols (e.g. **NO_FLOAT**, **NO_HMI**, **NO_ARGS**) and several new symbols have been added - one to disable plugin support altogether (**NO_PLUGINS**) and another for programs where the main C function never exits (**NO_EXIT**). All these symbols do is save some space in the final binary by removing unused program code (note that it is the programmer's responsibility to ensure that the code removed is not actually required!). For instance:

```
catalina hello_world.c -lci
     => code size of 6612 bytes
catalina hello_world.c -lci -C NO_ARGS -C NO_REBOOT -C NO_EXIT
     => code size of 6556 bytes
```

18. Fixed a race condition in some of the HMI plugins that meant the keyboard may not have been cleared correctly by **k_clear()** - occasionally a key would have been returned even if it had been sent BEFORE the keyboard was cleared.
19. A new serial plugin and HMI option have been added, based on the Full Duplex Serial driver. To use this driver as the HMI option, define the symbol TTY (i.e. by adding the option **-C TTY** to the command line). For example:

```
catalina hello_world.c -C TTY
```

    This plugin can also be used as a stand-alone serial port driver (e.g. in addition to an existing HMI option). Access functions are provided in the library libtty. This works in exactly the same way as the "4 port serial" plugin - all you need to do is link your program with libtty (i.e. by adding the option **-ltty** to the command line). For example:

```
catalina test_tty.c -ltty
```

Note that the default pin configuration for the TTY plugin may conflict with the default pin configurations used by the PC and PROPTERMINAL HMI options. If you use this HMI option, you can change the pins by editing the file *Extras.inc* - or if you do not need a HMI, simply define the symbol **NO_HMI** on the command line. For example:

```
catalina test_tty.c -C NO_HMI -ltty
```

The main advantage of using TTY HMI option over the PC HMI option is that it saves one cog. The main disadvantage is that it cannot be proxied (which only matters in a multi-CPU system).

20. Fixed a bug in libserial4 that could lead to corrupt output when the multi-threading kernel was in use. Added a more complex demo program to verify the bug is now fixed.
21. The inter-prop loaders used on multi-prop systems (e.g. TRIBLADEPROP and MORPHEUS) no longer displays status messages when loading a CPU with a screen attached (e.g. CPU #1 on TRIBLADEPROP, or CPU #2 on MORPHEUS). While this was useful as a visual check that the loader is in fact working, it limits the size of programs that can be loaded to those that fit below the buffers required for the display. So it is now not enabled by default. To restore the previous behavior, explicitly define the symbol DISPLAY_LOAD (using the option **-D DISPLAY_LOAD**) in the homespun command that compiles the relevant loader (i.e. *Generic_SIO_Loader_2.spin* and *Generic_SIO_Loader_3.spin*).
22. Fixed a bug in the proxy drivers - if the mouse or keyboard was disabled (via **NO_MOUSE** or **NO_KEYBOARD**) when using the proxy drivers it was possible that the mouse and/or keyboard buffers were still allocated - but at the same Hub RAM location as the screen buffers. This could lead to corrupt characters being displayed on the screen.
23. Fixed a problem when specifying the PROPTERMINAL HMI option in Code::Blocks (Code::Blocks was defining the symbol **PROPTERM**, not **PROPTERMINAL**, so the correct driver was not being included).
24. Added the ability to specify **NO_SCREEN** via a Code::Blocks HMI options.
25. The default FG and BG colors used in all the supported TV and VGA drivers are now specified in Catalina_Common.spin. This makes them much easier to configure, and they can be configured differently for each driver (e.g. TV, HIRES_TV, LORES_TV, VGA, HIRES_VGA, LORES_VGA etc). The default in all cases is set to White text on a Dark Blue background. Note that the way colors are specified differs between different driver types - refer to the low level drivers for more details (e.g. TV.spin, VGA.spin etc).

Release 3.5:

26. Fixed some typos in the Catalina Reference manual - thanks to Ray Rodrick.
27. Added new library functions to access more of the Propeller special registers (**frqa**,**frqb**,**ctra**,**ctrb**,**phsa**,**phsb**,**vscl**,**vcfg**) and also fixed some of the existing functions (**outa**,**outb**,**dira**,**dirb**) to prevent toggling outputs while updating their value - thanks to Ted Stefanik.

28. A function to create a directory has been added (i.e. _create_directory(), documented in **catalina_fs.h)**. The path to the new subdirectory must already exist - e.g. if you want to create *\dir1\dir2\dir3* then the directory *\dir1\dir2* must already exist. There is currently no way to delete a subdirectory.
29. A section on file system support, describing the various means of accessing SD Card based file systems (i.e. via the standard C "stdio" functions, or via managed or unmanaged Catalina file system functions) has been added to the Catalina Reference Manual.
30. The default platform (if none is specified on the command line) is now the CUSTOM platform, which comes preconfigured for a Propeller with a 5Mhz clock and not much else. The default HMI option for this platform is the PC HMI, so this platform is suitable for nearly ANY propeller with a 5Mhz crystal, including the Parallax QuickStart board or various Gadget Gangster boards. The CUSTOM platform is now preconfigured in ALL the target packages provided (i.e. target, basic and simple) and no longer requires any further editing for Propellers that use a 5Mhz clock, no XMM RAM, and only serial input and output capabilities.
31. Payload now has an interactive mode, enabled via a new **-i** switch. When enabled, the Payload program turns into an interactive terminal emulator immediately after the program has been loaded. Programs compiled with the PC HMI option will therefore be able to be more easily used, and no output will be lost while the user starts or enables their terminal emulator after downloading a new program.
32. Code::Blocks Tools menu has been updated to add a new tool called **Download to Hub RAM and Interact**, which enables the new "interactive" mode of the Payload program, acting as a terminal emulator.
33. The Code::Blocks Catalina project wizard has been enhanced to also allow the HMI option to be selected. The defaults have now changed - the CUSTOM platform is now the default, and the PC HMI option is now the default.
34. The Catalina QuickStart guide now uses the PC HMI as the default HMI option when building the first demo, and the new "interactive" mode of Payload. This means the demo should work on nearly any Propeller platform.
35. Fixed a bug in the standard C function **ungetc** that occurred when using the normal non-extended libraries (**-lc** or **-lci**). This bug also affected **scanf** and may have affected other stdio library functions. It did not affect programs that used the extended libraries (**-lcx** or **-lcix**)
36. Catalina now generates less Homespun messages on each compile. Most of the messages were not useful, so it now only prints a message to indicate which Spin modules it is including in the final executable (this is quite useful in verifying that the command-line options have been correctly specified). To generate all the previous information messages, use the **-v** switch.
37. Several minor changes have been made to the Catalina virtual machine which reduces the code size and improves the performance of all kernels slightly. For programs that do not require floating point, the speed improvements in the **ALTERNATE** kernel are quite significant. Note that to use the **ALTERNATE**

kernel for programs that don't need floating point support, the **NO_FLOAT** symbol should also be defined (to prevent Catalina including the floating point plugins that make up for the **ALTERNATE** kernel missing the "in-kernel" floating point capabilities). For example:

```
catalina hello_world.c -lci -C ALTERNATE -C NO_FLOAT
```

38. Addition of the new "service-oriented" registry. A full set of registry manipulation functions provided for use from Spin programs has been added. Catalina provides Spin functions to access the three different registry layers, but by default only the "layer 1" functions are included in full. If you need access to the full "layer 2" registry functions (i.e. the "cog-based" registry) from Spin, define the symbol **FULL_LAYER_2** on the command line. If you need access to the full "layer 3" registry functions from Spin (i.e. the new "service-based" registry), define the symbol **FULL_LAYER_3** on the command line. Access from C to all layers is always included.
39. More tidying up and simplification in various target files. The addition of the new "service-oriented" registry (in particular the way memory is allocated during startup) has significantly simplified the way plugins are set up and loaded from Spin.
40. Added new service-oriented registry functions to catalina_plugin.h. Also, for more consistency between C programs and Spin programs, the names of the C plugin types has been prefixed with **LMM_** (e.g. **HMI** -> **LMM_HMI**). Any C programs that use these names will need to be updated.
41. Renamed the **simple** target to **minimal**, since this is more descriptive of its intent.
42. Fixed a bug when using the external floating point plugins from multiple kernels executing C programs concurrently.
43. The new "service-oriented" registry incorporates automatic management of access to plugins from multiple programs. This is enabled automatically if the multithreaded kernel is used. You can enable it manually by defining the symbol **PROTECT_PLUGINS** on the command line. This would be required (for example) if you were using the default single-threaded kernel, but planned to dynamically load additional kernel cogs.
44. 19. Added a new command line option (**-C**) to define 'Catalina' symbols (i.e. symbols that affect the Spin target files). Previously this was done using **-D**, but **-D** is now reserved for defining C language symbols. The same change has been made to catbind (but note that Homespun still uses **-D**). When you define a Catalina symbol **XXX**, the compiler also defines the C symbol **__CATALINA_XXX** - this can be very handy for accessing the target configuration options from within C programs. So there are two ways to define symbols:

    -C is for 'Catalina' symbols (i.e. symbols that affect the Spin target)

    -D is for 'normal' symbols (i.e. that affect only the C program).

    For example

    ```
    catalina hello_world.c -lci -ltiny -C PC -D printf=tiny_printf
    ```

    This would define the symbol **PC** in the Spin target files, and the symbols **__CATALINA_PC** and **printf** in the C program.

45. Fixed a problem in the floating point plugins (*Float32_A_Plugin.spin* and *Float32_B_Plugin.spin*). There was an error in the **_FCmp** function - it worked ok internally, but was not returning the result correctly when called from outside the cog. This would not have caused a problem unless a program was manually invoking this request, since prior to Catalina 3.5, all kernels used a built-in version of **_FCmp** that worked correctly.

46. Added library libtiny, based on Ted Stefanik's amazing library. To use this library add **-ltiny** to your command line (in addition to the standard C library included by **-lc**, or **-lci** etc). For example:

```
catalina hello_world.c -lc -ltiny
```

    The tiny library provides smaller versions of the stdio functions **printf**, **scanf**, **sprintf** and **sscanf**, which are generally compatible with their stdio equivalents, but much smaller. For even smaller programs, there is also a REALLY small version of **printf**, (called **trivial_printf**) which can be enabled by using a command similar to the following:

```
catalina hello_world.c -lci -ltiny -C printf=trivial_printf
```

    See the header file *tinyio.h* for more details.

    Note that the libtiny library fixes a problem in libc and libci that does not support **sscanf**.

47. Added the 4 Port Serial plugin, and a new library to access it. To use this new plugin, simply add -lserial4 to the command line. For example:

```
catalina test_serial4.c -lci -C NO_HMI -lserial4
```

    See the file *catalina_serial4.h* for a definition of the functions provided to access the serial ports. To configure the serial ports, edit the *Extras.spin* file. A section on serial support has been added to the Catalina Reference manual.

48. It is no longer necessary to define the **THREADED** symbol on the command line to enable the use of the multithreading kernel. Instead, simply include the threads library - the multithreaded kernel will be selected automatically. For example:

```
catalina test_multiple_threads.c -lci -lthreads
```

49. Made the **PC** keyboard buffer larger (and configurable!). Thanks to Ray Rodrick for this enhancement.

50. Several command line symbols have been eliminated, since Catalina can infer what is required by the inclusion of various libraries. Such "inferences" are implemented in the various target files, and can be modified if required.

    The following "inferences" already existed in previous versions:

    -lma => automatically includes the "Float A" plugin

    -lmb => automatically includes the "Float A" & "Float B" plugins

    -lmci => automatically includes the "SD Card" plugin

    -lmcix => automatically includes the "SD Card" plugins

    The following new "inferences" have been added:

    -lthreads => automatically selects the threaded kernel

-ltiny => automatically renames printf, scanf, sprintf & sscanf

-lserial4 => automatically includes the "4 port serial" plugin

-lgraphics => automatically includes the "CGI" plugin

This has eliminated the need for the old **THREADED** and **GRAPHICS** symbols.

51. Added mouse driver to the C version of the Parallax graphics demo. Also, the batch file now only builds one version at a time, as it became too cumbersome for the batch file to figure out all the possible combinations when building various versions, so it now only builds one and you have to explicitly tell it the options you want.
52. Added a **DISABLE_REBOOT** symbol, which prevents a C program from rebooting automatically should it ever exit the main function. Instead, it just enters an infinite loop. Use it by defining the symbol on the command line. For example:

```
catalina hello_world.c -lc -C DISABLE_REBOOT
```

53. The CUSTOM platform in each of the target directories (target, basic, minimal) is now configured appropriately for a Parallax QuickStart board.
54. Updated the **basic** target package to include the new 4 port serial driver. Since the basic target package is intended for deeply embedded applications, it does not include any HMI plugins (including the normal serial **PC** HMI plugin) - but serial communications is quite a common requirement for embedded applications, and the 4 port serial driver is very appropriate.
55. The **PC** and **PROPTERMINAL** HMI options now use the value of **SIO_BAUD** defined in the platform definition file (e.g. *HYDRA_DEF.inc*) instead of being hardcoded at 115200 baud. All this does is make it simpler to change the baud rate if required, since it only needs to be changed in one place.
56. The default size for any XMM program is now 16Mb.

Release 3.4:

1. Fixed a bug in **setjmp()/longjmp()** implementation - these routines were not saving/restoring the registers, which meant register variables were being lost.
2. Fixed a bug in the code generator which meant that multiply, divide and mod operations were not being "spilled" properly in case another such operation was performed before the result was saved. This occurred most commonly in parameter lists that contained arithmetic expressions for more than one such parameter.
3. The Catalina "One-touch" installer for Windows now does not install the Catalina source code by default - it is now an option. This was done to improve the install speed on slower Windows machines.
4. Fixed error message detection for Homespun errors - some messages were not being correctly identified as errors.
5. Fixed an error in building the utilities for the SuperQuad – the script used tried to build them using the LARGE memory mode, but this gave an error

message about the mode being not supported. Now it builds them with the SMALL memory model (this makes no difference to boards that support both SMALL and LARGE).

6. Modified the *DracBlade_HMI.inc* file to allow TV HMI options, since some DracBlades now support TV output. However, the default is still HIRES_VGA, so for TV support you must specifically define the **TV** symbol (or **HIRES_TV** or **LORES_TV**).
7. The default size for all XMM programs is now 16M (16777216), so it is no longer necessary to specify the **-M** option in most circumstances (except when also using the **-e** option, or all **.eeprom** files will end up 16M in size).
8. Rewrote the LMM Support, XMM Support and EMM Support sections to try and clarify the different targets and loader options.
9. Fixed the demo build_all scripts to make the program names align with the code::blocks names for the same programs.
10. Modified the Windows "One touch" installer so that it will prompt before overwriting any modified versions of "Custom" configuration files – these are any files with a name like "*Custom_*.**" in the *basic*, *simple* or *target* folders - this ensures any custom modifications are not accidentally lost just because an upgrade to a new version of Catalina is installed. However, note that completely uninstalling Catalina ***will*** delete these files.

Release 3.3:

1. Support for FlashPoint XMM boards (SuperQuad and RamPage). Although they are both XMM boards that must be connected to a suitable propeller platform, each has been assigned its own platform symbol (**SUPERQUAD** and **RAMPAGE**). Or the necessary configuration details can be incorporated into the files for another platform (e,g. CUSTOM). Since they are quite similar, the options for both boards are defined in the configuration files:

   FlashPoint_DEF.inc
   FlashPoint_CFG.inc
   FlashPoint_HMI.inc
   FlashPoint_XMM.inc.

   See *FlashPoint_README.txt* for more details on how these files can be used.
2. Removed all platform-specific options from the plugins, and added a new CFG file for each platform (e.g. *Hydra_CFG.inc*) which contains all plugin configuration options for the platform. This makes it much easier to support new platforms without having to edit each plugin file.
3. Included the RTC (CLOCK) functionality in the SD plugin – now, if the SD plugin is loaded, the CLOCK plugin will not also be loaded – instead, the CLOCK code embedded in the SD plugin will be used. If the SD plugin is not loaded, the CLOCK plugin will still be loaded as usual. This saves a cog in many cases on large complex programs. This is all automatic - no user intervention is required.

4. Formalized the XMM API to include both Cache and Flash support. This was required since the quad-bit SPI used by the FlashPoint modules could not be accommodated in the existing Flash API (which could only cope with the single bit SPI chips used by the C3 and Morpheus platforms). Now all platforms use a common XMM API which is divided into three parts:

   the cached API, intended to be used only via the cache

   the direct API, intended for inclusion directly in the kernel

   the flash API, intended to be used when Flash support is required.

Release 3.2:

1. Fixed an error in the definition of **FLT_MAX**, which led to a compiler warning about floating point values overflowing.
2. Simplified the multi-cog demos by providing a precompiled version of the dynamic kernel which is automatically loaded when the new **_coginit_C()** library function is used to start a C function in a new cog.

   Similarly, simplified the spinc demos by adding a new **_coginit_Spin()** function to the library that can be used to start a Spin program in a new cog.

   Now there are the following functions for starting a new cog:

   in *catalina_cog.h*:

   _coginit : start an arbitrary binary program in a cog

   _coginit_C : start a C void function in a dynamic kernel cog

   _coginit_Spin : start a Spin program in a spin interpreter cog

   in *thread_utilities.h*:

   _thread_cog : start a C main function in a multi-threaded kernel cog
3. Tidied up all the demo programs to us an ANSI compliant main program.
4. Increase the maximum size of the command line in the **catalina** and **catbind** utilities. This is necessary because of the length of some of the Code::Blocks build commands.
5. Add detection of Homespun "Item has already been added" error (when a symbol has been multiply defined) to CodeBlocks.
6. Tidied up the **spinc** demos and added the **-f** flag to **spinc** utility. Also, created scripts *spinc_to_file* for use within Code::Blocks pre/post build commands (since redirection is not supported within Code::Blocks).
7. Fix some errors in the Catalina Code::Blocks compiler plugin - the **libcix** and some of the proxy driver options generated invalid Catalina options. Also, removed the **RESERVE_COG** option and added a "suppress warnings" option.
8. Added the build_catalyst script to allow building Catalyst from within Code::Blocks. If **MinGW** and **MSYS** are installed, this will build both Catalyst and all the demos. If not, the script will build the core of Catalyst, but the demos will have to be built from within Code::Blocks.

9. Added work spaces and project for building all the Catalina and Catalyst demo programs from within Code::Blocks.
10. Fixed a race condition in the initialization of the RTC (**CLOCK**) plugin.

    Payload now detects the ports to use automatically even on the **HYDRA** and **HYBRID** platforms when the special mouse port cable must be used.

Release 3.1:

1. Fixed a bug in the **SMALL** memory model on the C3 (same bug as was fixed in 3.0.3 - the fix was lost in release 3.0.4!).
2. Simplified the "standard" target package. Turned all the plugin selection include files into SPIN objects to make the logic of the plugin load process easier to understand. Removed the include files containing the kernel and loader selection logic, as they hid some of the differences between targets, which also made the logic harder to understand.
3. Removed the duplication of HMI plugins - now there is only one HMI plugin for each screen type, and not a separate for omitting the mouse and/or keyboard drivers.
4. The POD assembly language debugger is now enabled by compiling using the target "pod" rather than "debug" - this is to avoid confusion between the blackcat/blackbox and pod debug support files. For example, to use pod to debug hello_world.c, you would now use a command like:

   ```
   catalina hello_world.c -lci -t pod -D NO_ARGS -D NO_FLOAT
   ```

   Note that POD still requires the use of PropTerm, with the terminal set to 40 characters by 30 rows. The program to be debugged should be uploaded using PropTerminal, and should not itself use the serial port.
5. Added a "basic" target package. The basic package supports only one platform, and no HMI drivers. It is intended for very deeply embedded applications where HMI drivers are not required, or a user prefers to add their own HMI drivers.
6. Removed the **RESERVE_COG** support. This also complicated the logic in many target files, and the main purpose of this functionality (to prevent a cog being initialized during the load process so that SDRAM could be used as XMM RAM) was implemented in the Caching XMM driver instead.
7. Added some notes about MinGW and Windows 7 in the *README.Win7* file. For various reasons, Windows 7 does not socialize well with MinGW.
8. Add ".exe" to the references to catalina and catbind in the various Catalyst demo program makefiles. This seems to be required to allow MinGW to execute the programs correctly under Windows 7. This is a known issue, and is probably a Windows security "feature".
9. The MORPHEUS demo program to test the file system was incorrectly being built for CPU #2 instead of CPU #1 in the tutorial.
10. Fixed a bug on Linux where the **libcix** library was not being built correctly.
11. Renamed the *custom* target directory to *simple* (and moved *custom_demos* to *demos\simple*) to avoid any confusion between the *custom* target package

and the **CUSTOM** target (which is a target within the standard target package). The two have nothing to do with each other!

12. Moved the RAM test programs to the *utilities* directory. They can be built using the command **build_ram_test** in that directory.

Release 3.0.4:

1. Added improved **spinc** utility, which allows the execution of Spin programs as well as just PASM programs. For details on the many new **spinc** options, execute the command

```
spinc -h
```

Release 3.0.3:

1. Completely New Code::Blocks integration specifically for Code::Blocks 10.05. A new Compiler plugin that supports Catalina has been included, plus a new Catalina Project Wizard, and wrapper functions to allow Catalina utilities to be invoked from the Code::Blocks tools menu.
2. Added the "build_utilities" batch file, which interactively guides users through the options required when building the "utilities" folder.
3. Make catbind use the same algorithm as LCC to determine a directory to use for temporary files. Like LCC, the default directory can be overridden by setting the **CATALINA_TEMPDIR** environment variable. This should make it possible to use Catalina when logged in as a user without Administrative rights under Windows 7. However, it is still necessary to disable the VirtualCopy feature of Windows 7 - see the file *README.Win7*
4. Fix a problem in *catalina_env.bat* under Windows 7.
5. Minor changes to some build_all scripts and makefiles. Catalina now assumes MinGW and MSYS are installed, and will use the versions of programs (e.g. make, bison, flex) that come with MinGW. Make sure you set your path variable appropriately when building Catalina - for example, if MinGW and MSYS are installed in "C:\MinGW" you would need to do the following:

```
set PATH=C:\MinGW\bin;C:\MinGW\mingw32\bin;C:\MinGW\msys\1.0\bin;%PATH%
```

6. Fixed a problem with payload failing to load *XMM.binary* from the bin directory.
7. Change in terminology - payload now says "download" instead of "upload"
8. Fixed a bug that occurred when using the **-lmb** option which left the symbol Float_B undefined.
9. Fixed a bug with the **SMALL** memory model on FLASH platforms (such as the C3) that led to programs not having their data segments incorrectly initialized.
10. Blackbox now automatically finds the debug port if none is specified.

Release 3.0.2:

1. Fix a bug in the plugin search algorithm used in the XMM kernel which meant the search stopped when the plugin type was 8 instead of when it reached cog number 8! This meant that plugin type 8 (allocated to the DUM plugin) could never found in XMM programs.

2. Fixed some typos in the definition of the **waitpne** and **waitpeq** macros in the header file *catalina_icc.h*

Release 3.0.1:

1. Fix a bug in the HUB Flash loader (used on the C3 and Morpheus) which meant the read/write sectors were not being correctly copied from Flash to SPI RAM. This caused mysterious failures such as the SD card file system not working correctly.
2. The default behavior when programming SPI FLASH (e.g. on the C3 and the Morpheus platforms) is now to erase each 4k block just before programming (instead of initially doing a full chip erase). The previous behavior can be reinstated by defining the symbol **CHIP_ERASE** when compiling (e.g. when compiling the utilities or Catalyst folders). One result of this is that when using Payload to load a program on the C3, it is no longer necessary to extend the payload timeout by adding an option like **-t 1000** to the command. Another benefit is that loading programs into SPI FLASH is faster when using both Payload and the Catalyst loader.
3. Added **ERASE_CHECK** and **WRITE_CHECK** options to the caching SPI driver. These options can help improve reliability when programming the SPI FLASH (e.g. on the C3 or Morpheus). Note that **ERASE_CHECK** is only applied to the whole flash chip erase, not the block erase. Only the **WRITE_CHECK** is enabled by default, since it does not take much time (the erase check, on the other hand, can take around 15 seconds!). To enable the erase check, define the symbol **ERASE_CHECK** when compiling (e.g. when compiling the utilities or Catalyst folders). Doing so may require even longer timeouts when using Payload (e.g. **-t 4000**).
4. The library functions **_register_plugin** and **_unregister_plugin** now mask the cog id to 3 bits, to eliminate the chance of overwriting memory beyond the end of the registry. However, note that this does mean specifying something like cog 9 will actually register or unregister cog 1!
5. Add a **ZERO_RAM** option to the RAM test program. Compiling the RAM test with the symbol **ZERO_RAM** defined will cause the RAM test to zero all read/write XMM RAM before performing the test (and it will also disable the initial display of the XMM RAM). The ability to zero all XMM RAM (which can be done simply by starting the RAM test but not proceeding to perform any tests) is useful in diagnosing some program bugs or load problems.
6. Added an option to the RAM tester to display FLASH only on startup (define the symbol **PRINT_FLASH_ONLY**), and another to print continuously until a key is pressed (define the symbol **PRINT_CONTINUOUS**). This is handy for dumping the entire contents of Flash (save it by using a PC terminal emulator and saving the terminal buffer contents) when debugging loader problems.
7. Added C3 as a platform (and also various new options) to the messages displayed by various *build_all* batch files.
8. Added a new script to the bin directory (Set_Linux_Permissions) to set the permissions of all Catalina files in a Linux installation.

Release 3.0:

1. Added functionality to payload to also look in the Catalina binary directory for files to load into the propeller. Also, upgraded the build scripts in the utilities directory to copy the binary files generated to the binary directory. The effect of this is that it is no longer necessary to specify the full path name to the **XMM.binary** loader (or the other payload utilities such as the mouse port loaders) or copy them to your current working directory each time you need them. For example, to upload an XMM program, you can now simply say (from any directory):

   ```
   payload XMM program
   ```

   Just make sure you have previously built the utilities for the correct platform!
2. Added **-m** (max retries) option to payload. This sets the number of times operations are retried. The default value is 5.
3. Fixed a bug with **setjmp**/**longjmp**. In LMM and XMM small mode these functions worked on some platforms but not others. In XMM LARGE mode these functions failed on all platforms.
4. Added a CACHED XMM driver for use with SPI RAM and SPI FLASH. The caching XMM driver can be used on any platform by defining one of the following symbols on the command line:

   CACHED_1K use a 1k cache

   CACHED_2K use a 2k cache

   CACHED_4K use a 4k cache

   CACHED_8K use a 8k cache

   CACHED use a default size (8K) cache

   For example:

   ```
   catalina hello_world.c -lc -D CACHED
   ```

   Note that the caching XMM driver requires sufficient free hub memory according to the size of the cache, and also an extra cog. Also, the caching XMM driver is only generally faster on platforms that use SPI RAM or SPI FLASH - on other platforms the caching driver may only be slightly faster, or may even be substantially slower! The caching XMM driver is primarily intended for platforms such as the C3 or Morpheus CPU 2, which only have serial SPI RAM or SPI FLASH available for use as XMM RAM.

   Note that when a program which uses the caching driver is loaded by Catalyst, and Catalyst itself has been compiled to use the cache, then the cache size used for Catalyst **MUST BE THE SAME** as that used by the program being run. However, Unless dynamic RAM is in use (which requires constant refresh to retain it's contents) then it is perfectly ok for Catalyst to to be compiled to WITHOUT the caching XMM driver but still be used to load programs which do - it is only when both Catalyst and the program to be loaded use the cache that **IT MUST BE THE SAME SIZE!** But if Catalyst has been compiled to use the cache, then so must all the XMM programs that are to be loaded.

   To summarize, here are the possibilities (assuming static RAM):

Catalyst Uses Cache	XMM Applications
Yes	MUST use same size cache
No	May use a cache of any size, or no cache

When dynamic RAM is in use then both Catalyst and all XMM applications MUST use a cache, and this cache MUST be the same size.

Failure to abide by these rules may result in programs that appear to load successfully, but fail to run correctly.

5. Added two new layouts intended for use on platforms with SPI FLASH (e.g. on the C3 and the Morpheus platform) - layout 3 and layout 4. Note that both of these layouts require the use of the CACHED XMM driver.

   Layout 3 - similar to layout 5, but the read-only segments (code and cnst) are in SPI FLASH, and the read/write segments (init & data) are in SPI RAM.

   Layout 4 - similar to layout 2, but the read-only code segment is in SPI Flash, and all the data segments (cnst, init & data) are in Hub RAM (SPI RAM is not used in this mode).

   These layouts can be modified slightly by using the **-P** and **-R** command line options:

   -R sets the base address of the Read-Only segments. It should be set to the base address of the SPI Flash.

   -P sets the base of the Read/Write segments. It should be set to the base address of the SPI RAM.

6. Added a new **Flash_Boot** utility, to boot programs already loaded into SPI Flash. Note that when compiling the utilities, you should now specify ALL the plugins you want - only plugins specified will be loaded by the **Flash_Boot** utility. For example, if you want the **Flash_Boot** program to load an SD Card driver, a **CLOCK** driver and use the **HIRES_vGA** HMI but not load a mouse, you would compile the utilities using a command like:

   ```
   build_all C3 CACHED SD CLOCK HIRES_VGA NO_MOUSE
   ```

   To run a program loaded into SPI Flash on Propeller reset, you can program the **Flash_Boot** binary into EEPROM. You can also recompile the **Flash_Boot** program with different drivers - this might (for example) enable you to run the program in SPI Flash with either a TV driver or a VGA driver.

   Note that the program in Flash will NOT be aware of whether or not the caching driver is in use, or of the size of the cache. This means that the same program can be executed with different cache options.

7. Removed the **Catalina_XMM_SD_Loader** (& the **Generic_SD_Loader**). These were deprecated in previous releases, and have now been completely replaced by the **Catalyst_SMM_SD_Loader** (& **Catalyst**).

8. Tidied up the XMM routines in the Kernel - they were not working exactly as the documentation indicated - now they do.

9. Added a Flash test to the **RAM_Test** program. This is currently only enabled for the CACHED XMM driver. It simply displays the EEPROM, erases it and

displays it again, then writes to it and displays it again. Also tidied up the programs user interface.

10. All HMI plugins now default to ANSI complaint interpretation of the control characters BS, HT, VT, CR, LF & FF (BEL is still ignored) on output. If you have a program that depends on the prior (i.e. non-ANSI) behavior, you can define the symbol NON_ANSI_HMI. This can be done on the command line at run time. You may also need to define the symbol **CR_ON_LF** (see below).

    The default behavior of all HMI plugins is now to translate a CR on input to an LF. Previously, this was done in the C library, but this meant it could not be disabled easily. Now it can be disabled in the HMI plugins themselves at compile time by defining the symbol **NO_CR_TO_LF** on the command line.

    Finally, some Windows terminal emulators need LF in the output translated to CRLF - this can now enabled at compile time by defining the symbol **CR_ON_LF** on the command line.

    For example, to use the othello program with a Windows terminal emulator that expects to see CRLF as the line terminator (and which cannot be configured to do this automatically) you can use a command like:

    ```
    catalina -lci othello.c -D PC -D CR_ON_LF
    ```

11. Disable Catalyst banner output if executing the 'autoexec' command.
12. Register all driver cogs. Previously the main HMI cog was registered, but the underlying driver cogs for keyboard, mouse and screen, or the proxy client driver, were not individually registered. Now they are.
13. Added some macros to make accessing the registry easier. See the include file *catalina_plugin.h* for details, or the demo program *test_plugin_names.c* for an example of their use.
14. Fixed a bug in the initialization of the SD card driver. Now it retries if the first initialization did not succeed. Affected some platforms but not others - possibly depending on the type of SC card used.
15. Fixed a bug in Catalyst that meant some commands were not being passed their arguments correctly.
16. Under DOS, Catalyst will now not build unless "make" is in the path. Previously, Catalyst itself would build but many of the demo programs would not (but with little indication of failure).
17. The Generic SIO Loaders now cannot display progress on the screen of the CPU doing the loading (due to lack of space). Now the progress can only displayed on the screen of the CPU being loaded - this is enabled by default when loading programs to Morpheus CPU 2 and TriBladeProp CPU 1. This can be disabled by removing the **-D DISPLAY_LOAD** command line option in the *build_all* script in the utilities folder. Disabling it enables slightly larger programs to be loaded, and may also speed up the load time - but there is no visible indication that the load is progressing.
18. Fixed a bug where the keyboard was not initialized prior to program start. This could result in one or more spurious characters being received on startup. This was particularly evident when using the proxy drivers on a multi-CPU Propeller system (such as the TriBladeProp).

19. Fixed several problems in the tutorial and programs designed to demonstrate the proxy drivers on both the TriBladeProp and the Morpheus platform. The proxy driver tutorial programs now work as the stated in the **Getting Started with Catalina** document.
20. Now accept defining the symbols **TINY**, **SMALL** and **LARGE** on the command line to specify the memory model. In conjunction with the existing symbols **SDCARD**, **EEPROM** and **FLASH**, these symbols now allow all the supported layouts to be specified as an alternative to using the **–x** option (which is retained as a shortcut, and also for compatibility purposes).

Release 2.9:

1. Upgraded to Homespun version 0.30 - this means there is no longer an 'input' subdirectory in the target directory, since files no longer need to be preprocessed to include the symbol definitions. Also, the new include file capability means a lot of common code has been factored out and is now specified in various '.inc' files. Because preprocessing is no longer required, some of the command line options (e.g. the catbind '-p' option) are no longer required. The end result is a significantly faster, cleaner and simpler compilation process.
2. Added a section for to **Catalina_Common.spin** for the ASC platform - thanks to Martin Hodge. To use it, specify -D ASC on the command line.
3. Added statistics (segment and file sizes) generation. This prints the size of each C segment (code, cnst, init, data) as well as the final file size (which will usually be larger than the sum of all segments as it will also include non-C code such as plugins). The statistics generation can be disabled by the -k command line option.
4. Added a new library variant - **libcix** - which can be used by specifying -lcix on the command line. This is an integer-only version of the extended library (libcx) - i.e. it includes fill file system support, but excludes any floating point support. This can reduce the size of programs that use the extended library by around 8kb.
5. Added **NO_FLOAT** as a synonym for the existing command line symbol NO_FP. When defined on the command line, NO_FLOAT and NO_FP both do the same - i.e. they prevent the loading of any floating point plugins. For example, when the XMM kernel is used, it normally loads the floating point plugin Float32_A since the XMM kernel includes no built-in floating point support (due to lack of space). However, if your program does not use any floating point, this is unnecessary and wastes a cog. To prevent this, you would use a command such as:

```
catalina hello_world.c -x5 -lci -D NO_FLOAT
```

6. Refactored code from the various targets into common include files. Now there is only one copy of many of the complex sections of code, which simplifies maintenance. For example, all the XMM access code now exists only in the file **XMM.inc**, and is included by other files as needed. The following include files contain code shared between multiple target files or utilities:

Constants.inc code to set constants for the Proxy and HMI plugins

FP.inc	code to include the appropriate Floating Point plugins.
HMI.inc	code to include the appropriate HMI plugin.
Kernel.inc	code to include the appropriate kernel.
Loader.inc	code to include the appropriate loader.
Other.inc	Code to include the appropriate other plugins.
XMM.inc	PASM code for XMM access (all platforms)

7. Added auto-execute processing to Catalyst. To auto-execute a command on boot, a file with the name specified by AUTOFILE (default is "AUTOEXEC.TXT") must be created in the root directory. If the file exists, it is read on startup, and the command it contains (up to the first zero terminator, or end of line character, or EOF) is executed. This command can be any valid Catalyst command, including the invocation of an external LMM, SMM or XMM program, and can include parameters to be passed to the program.
8. Added execution from SPI SRAM and FLASH on the Morpheus (CPU #1) and also on the C3. Added support for the –x3 layout and the –R option to allow read/write segments to be loaded into SPI RAM and read only segments into SPI Flash.

Release 2.8:

1. Added the **libthreads** Multithreading library. See the Catalina Reference Manual, or the descriptions of the threading functions in the header file *catalina_threads.h* for details.
2. Added the **libgraphics** Graphics library. See the Catalina Reference Manual or the descriptions of the threading functions in the header file *catalina_graphics.h* for details.
3. Added a NO_ARGS command line option. NO_ARGS prevents the inclusion of the **CogStore** object, and instead includes a simpler **CogStopper** object that will terminate cog 7 if CogStore is running in it. This is done to save space (and cogs) for programs that do not accept any arguments - even if the CogStore is loaded by the Catalyst program loader.
4. Better granularity of malloc/realloc/calloc/free. For example, now the code for realloc() and calloc() are not included unless specifically used by the program (previously they were always included if malloc() was included).
5. Better documentation about installing **Code::Blocks** if Catalina is not installed in the default location.
6. Added a new SMM memory/load model. This is actually a new two-phase loader similar to the EMM memory/load model, but intended for use on platforms that have an SD card. It allows C programs to include up to 31kb of application code, by first loading the drivers, then the application code, and lastly the kernel. The first 32kb of the file is the program that loads the drivers (only 31kb can be used), the second 32kb is the application program (again, only 31kb can be used) and the last 4kb - starting at the64kb boundary - must be the kernel. Thus all SMM programs are exactly 66kb (67,584 bytes) in size. It is used by defining the symbol **SDCARD** on the command line.

7. Catalyst now recognizes **.lmm**, **.smm** and **.xmm** as suffixes for Catalina program files (in addition to the normal **.bin**). The most important one is the ".smm" suffix, since Catalina must know that these functions do not get loaded into XMM RAM, even though they are larger than 32kb.
8. Minor change to the library - all I/O now uses the functions low-level function catalina_getc() and catalina_putc() for character I/O to stdin, stdout and stderr. This means that these routines can easily be replaced with customized routines if required. Previously, these routines were only used in the standard C library (libc) and the integer library (libci) but were not used in the extended library (libcx).
9. Added *catalina_fs.h* which describes low level "managed" and "unmanaged" file access routines. The "managed" routines are the traditional Unix low level routines, such as _open() and _close(). The "unmanaged" routines are similar, but have the advantage that they do not use malloc() or free() internally - instead, the caller passes these functions a block of memory for them to use. This makes the "unmanaged" routines smaller and also more suitable for use in an embedded environment. See *catalina_fs.h* for more details.
10. Fixed a bug in the code generator, when passing a constant to a function, and the constant was < 0 or > 512.
11. The _locate_registry() function was dependent on _registry() but this was not specified in the function - this means the symbol C__registry would be reported as undefined (unless the _registry() function was being imported by some other function).
12. Added the **RESERVE_COG** option to all loaders and targets. This option allows a cog to be loaded at the start of the load process (e.g. before any XMM RAM is accessed) and this cog will not be disturbed during the remainder of the load process. This is intended to allow things such as a dynamic RAM refresh cog to be loaded and left running throughout the process of loading and starting a C program. The example program provided (**Catalina_Reserved_Cog.spin**) simply toggles an I/O pin as proof that it is started and left running. The reserved cog is currently set to be a single cog (cog 6). If more than one reserved cog is required, this can be set in **Catalina_Common_Input.spin** by reducing the value of LASTCOG. Note that it is the responsibility of the user (or the target) to ensure that only cogs 1 to 5 are loaded (e.g. by plugins) during the load process if the RESERVE_COG option is specified. Also note that (depending on what the reserved cog actually does) the RESERVE_COG may need to be specified on ALL loaders and also on the various Catalina utility programs - e.g. for the HYBRID:

```
cd catalyst
build_all HYBRID RESERVE_COG
cd ..\utilities
build_all HYBRID RESERVE_COG
cd ..\ram_test
build_all HYBRID RESERVE_COG
cd ..\demos
build_all HYBRID RESERVE_COG
... etc ...
```

See the file **Catalina_Reserved_Cog.spin** for more details.

13. Fixed a few problems with the batch files, and also a problem with the serial loader on multi-prop platforms (TRIBLADEPROP, MORPHEUS) – support for serial load between cogs was accidentally disabled in the last release.

Release 2.7:

1. Some changes to the code generator, mainly to fix a problem caused when LCC's built-in "simplifier" removes common sub-expressions without checking whether they are re-used later (!).
2. An update to the *propeller_icc.h* include file to fix an error in the **msleep** macro, and also to make it unnecessary to edit the file to switch between the Catalina compiler and the ICC compiler (this is now detected automatically).
3. Fixed a bug with the PropTerminal HMI failing to detect the PC mouse - this driver should always assume the mouse is present.
4. Fixed a bug with the LCC preprocessor (**cpp**). If a parameter passed to a **#define** macro was in fact the name of another macro, then the preprocessor could enter an infinite loop, eventually crashing when it ran out of memory.
5. Default is now to use **XTAL2** on Hybrid.
6. Fixed a problem with the LCC **-l** command line option (affected Linux only).

Release 2.6:

1. Calls to floating point functions were left in the **t_printf** function in the **libci** variant of the standard library - this led to programs that would not link because some of the routines referenced **t_printf** were not present in the **libci** variant of the library.
2. Fixed some errors in the targets if various HMI combinations were used - e.g. if the **LORES_TV**, **NO_MOUSE** and **NO_KEYBOARD** options were specified the targets incorrectly threw an error indicating that the HMI options were not supported. Similarly for **HIRES_VGA**, **NO_MOUSE** and **NO_KEYBOARD**.
3. Fixed some errors in **_lockset()** and **_lockclr()** which meant they were not correctly returning the previous state of the lock. Also, **_locknew()** now returns -1 on failure (for consistency with other cog functions)
4. Added the **CUSTOM** platform. Specifying **CUSTOM** will always return an error until it is configured by editing **Catalina_Common_Input.spin**. By default, the **CUSTOM** target is presumed to support all HMI options and an SD card, but only one CPU and no XMM. All the build scripts and utility programs also understand the **CUSTOM** target.
5. Fixed an error in the **ALTERNATE LMM** kernel. This kernel was missing the **Entry_Addr** method required by the EMM targets (i.e. if the **-D ALTERNATE** and **-x1** command line options are specified).
6. Added the **-O** command line option to **catalina** and **catbind** to invoke the new Catalina Code Optimizer (However, note that the Catalina Code Optimizer is not included with the free version of Catalina).

7. Fixed an error with variadic functions declared using the pre-ANSI method (as specified in the *varargs.h* include file) instead of the ANSI method (as specified in the *stdargs.h* include file).
8. Some of the test programs did not pause before exiting. This was ok prior to release 2.5 since Catalina programs went into an infinite loop on exit from the main loop, but since release 2.5 the Propeller resets on exit from the main loop - this meant the output of some of the test programs was lost before it could be read.
9. Implemented functions to access the registry from within C programs. The following functions are now defined in *catalina_plugin.h*:
   - **_register()** : return address of registry
   - **_register_plugin(cog, type)** : register a cog as a plugin
   - **_unregister_plugin(cog)** : unregister a cog as a plugin
   - **_locate_plugin(type)** : find the cog of a plugin type (-1 if not found)
   - **_request_staus(cog)** : return the status of the last plugin request
10. Fixed an error in the **_clockinit()** function, which was not saving the new clock mode in byte 4 of hub RAM.
11. The cog functions **_dira()**, **_dirb()**, **_outa()**, **_outb()** now each return the original value of the respective register, this allows them to be used to retrieve the current value without changing it (by specifying zero for the mask and new value parameters).
12. Added a new dynamic LMM kernel, which can be loaded into another cog to execute C functions concurrently with the main kernel. Examples of using the new kernel are provided in the *demos\multicog* directory.
13. Added the **spinc** utility program, and demos on how to use it (in directory *demos\spinc*).
14. Significant updates to the Catalina Reference Manual, including details on all the HMI, cog and plugin support library functions.
15. Added the ability to "read" from **&function** or **&variable** in **blackbox** (which reads from the address of the function or variable specified). Also, a count has been added, to simplify reading multiple consecutive locations.
16. Fixed an undefined symbol (**ccheck**) that occurred in various targets when using proxy drivers for the screen but not the keyboard.
17. Example **xmm_payload** scripts included to simplify loading XMM programs when using the payload loader.
18. Updated Catalina Reference Manual note about code sizes, showing that a **hello_world** type program which initially compiles to 23kb can be modified slightly so that it requires under 1kb of Hub RAM.

Release 2.5:

1. Catalyst has been added. It is a more sophisticated version of the original Generic XMM SD Loader (which still exists). See the Catalyst documentation for details.
2. The Generic XMM SD Loader now implements a simple Catalyst-compatible command interpreter, with a couple of simple built-in commands - e.g. DIR. Also, the default extension is assumed to be ".BIN" if none is entered.
3. Both Catalyst and the Generic XMM SD Loader can now pass arguments to both Catalina C programs, and SPIN programs that use the Catalyst_Arguments.spin module provided.
4. When a C program ether calls exit() or returns from the main() function, it now reboots the Propeller. Any exit value specified is currently lost.
5. Fixed a bug in the setting up of the initial frame in main() which would have occurred if -g3 was specified.
6. Fixed a bug in the SIO loader that may have affected programs loaded with a different clock speed to the default clock speed for the platform.
7. Fixed a bug in the file system support, that meant opening an existing file for write did not truncate the file first - so if the new version of the file was smaller than the old one, junk was be left at the end of the file. This was evident when using xvi and deleting a line of text in a file.
8. Fixed some bugs in BlackBox that led to parse failures when loading a dbg file.
9. Increased the size of the BlackBox parse stack - required to parse very large dbg files (e.g. the one produced when debugging xvi!).
10. Fixed a bug in BlackBox when reading strings from XMM RAM.
11. Fixed a bug in BlackBox when printing the call stack when one .c file '#includes' another .c file
12. Tidied up the selection of CPU and HMI options - now Catalina correctly detects all invalid combinations (e.g. specifying TV on the DRACBLADE, or CPU_3 on MORPHEUS, or VGA on the TRIBLADEPROP CPU_3).
13. Added RAMBLADE support. Currently, the Catalina_Common_Input.SPIN file assumes a 6.25MHz crystal on the RAMBLADE, for a frequency of 100MHz. This appears to be the maximum frequency at which the RAMBLADE can reliably use both the SD card and the XMM RAM. If your RAMBLADE can be overclocked to a higher frequency, you can change the clock definitions in this file.
14. Inclusion of the C99 standard include files stdint.h and stdtype.h, and the supporting library functions. These include files define types such as int8_t and uint32_t that will be familiar to many C programmers.
15. Payload now supports '-e' to load a program into EEPROM. Note that -e can only be used to load SPIN or LMM programs into the first 32kb of the attached EEPROM - it does not currently support loading EMM programs into larger EEPROMs - this must still be done using external tools, such as the Hydra Asset Manager.

16. Payload now prints an error message if loading a program, programming the EEPROM, or verifying the EEPROM fails.
17. Tidied up all the command scripts (typically called 'build_all') to work more consistently - e.g they all now warn if the setting of CATALINA_DEFINE would conflict with the parameters specified on the command line, and will also warn if they need to know the location of the Catalina installation but cannot find it.
18. The Super Start Trek demo has now moved to the Catalyst folder, along with the JZIP, Dumbo Basic, XVI and Pascal P5 compiler (which has replaced the previous P4 compiler). These programs are no longer regarded as 'curiosities'.
19. The Catalina binder program has been renamed from 'bind' to 'catbind'. The old name ('bind') conflicted with another program on Linux, and on Windows using the word 'bind' in a Makefile also causes make to fail mysteriously.
20. Fixed a problem with the 'modf' function, which led (amongst other things) to slightly inaccurate floating point values (when printed with printf).
21. Fixed a few minor problems in BlackBox - now 'array' and 'func' are valid variable names.
22. Fixed all relative references in batch files and scripts. These made the scripts fail if they were copied to another folder. Now all the scripts that need to refer to absolute path names check the value of the LCCDIR environment variable. If that variable is not set then they use the fixed default paths ("C:\Program Files\Catalina" or "/usr/local/lib/catalina").
23. The Generic SD Card Loader now uses FATEngine instead of FSRW. There should be no difference from a user perspective. This was done because Catalyst required the more sophisticated file processing available with FATEngine.
24. Some minor speed improvements to all LMM and XMM targets, plus inclusion of Pullmoll's improved signed integer division code.
25. Move debug vectors out of special register space and into normal cog RAM.

Release 2.4:

1. Fixed a problem with include paths under Windows that contained a colon (the path would be split at the colon unless a ';' was not included).
2. Fixed Code::Blocks configuration for Linux - a separate set of Code::Blocks configuration data is now provided for Linux, along with a separate example workspace.
3. Fixed a bug in the binder that would cause it to fail to open the output file if the filename was quoted (e.g. because it contained a space).
4. Added debug options (-g and -g3) for generating debugging data. The compiler will generate a '.debug' file for each file compiled, and then use the new catdbgfilegen program to produce a consolidated .dbg file. Also, a listing file is now always produced when the debug option is specified. The difference between -g and -g3 is that -g3 disables some compiler optimizations to assist debugging - for example, it forces all functions to have a stack frame even if they don't normally need one - this allows program

function calls to be traced more easily, but can also make the program slightly larger and slower to execute..

5. Support for the BlackCat and BlackBox debuggers has been added - this is enabled by using the -g or –g3 command line options (described above). Note that when using Code::Blocks, the -g (or –g3) option also has to be specified on the list of linker options - it is not sufficient to merely set the compiler option. This is done automatically on Debug targets if you use the Catalina template provided.
6. Fixed a bug in the Generic SD and SIO Loaders - now initializes Hub RAM before starting programs, and also checks for clock speed changes.
7. Support has been added for the DracBlade.
8. Fixed a bug in the HiRes_VGA_No_Mouse driver.
9. The default debug demo is now a version of 'hello_world' (since Othello cannot be built on platforms that use VGA - it requires too much memory).
10. A symbol is now defined to indicate that the SMALL memory mode is in use. The symbol SMALL can be used in the SPIN initialization code, and the C symbol __CATALINA_SMALL can be used within C programs (this complements the corresponding symbols LARGE and __CATALINA_LARGE that are defined when the LARGE memory model is in use - if neither of thise symbols is defined then the TINY memory model is in use).
11. Fixed up the 'build_all' script in the demo directory to either accept command line parameters specifying the build options, or use the current value in the CATALINA_DEFINE variable (or both, if only one parameter is specified and it is identical with CATALINA_DEFINE).
12. Fixed some of the build scripts to accept multiple parameters, such as the ram test program - this allows building for TRIBLADEPROP CPU_1, or to include -g on compile of various demo programs. Also modified the script in the demo directory to allow the use of either command line parameters or the CATALINA_DEFINE environment variable - now the script only complains if both are set but are not identical.
13. Fixed a bug in the XMM support for the TRIBLADEPROP CPU_1 which could lead to corrupted writes to XMM RAM.
14. Implemented clockmode(), clockfreq(), and clockinit() functions for managing the propeller clock - they are described in "catalina_cog.h".
15. Added Catalina Payload. This program can be used to load programs from the PC directly into a Prop - including SPIN/PASM, LMM or XMM programs. On platforms that use the HX512, it supports using another serial port (e.g. the mouse port) to load XMM programs, since the normal serial port on pins 30/31 cannot be used once the XMM card is initialized.
16. Fixed various problems preventing Catalina being used by multiple users, or by non-root or non-Administrator users. The reason for this was that Catalina was not cleaning up the 'target' directory after each compile, which meant the next compile (by another user) would fail. Now, Catalina correctly cleans up after itself. If for some reason it does not (e.g. the compiler crashes) there is a 'catalina_clean' script provided which can be used to clean out all the

intermediate and generated files from the target directory - this script must be run as root or Administrator. Also, to enable multiple users, the 'target' directory must be writable by all users. Problems with this usually result in error messages such as 'cannot open output file' - if you get these, run the clean script. Note that it is still NOT possible for two compilations to run concurrently - if two different users try it, one will get an error message. However, no error message will be issued if the same user performs two parallel compilations - but the results will probably be incorrect. This means Catalina is not currently suitable for use with 'make' type programs that perform parallel compilations.

17. Fixed a problem with Catalina being unable to create temporary files under Windows 7 or Vista when used by a user other than an Administrator.
18. Catalina now sends error messages to ‘stderr’ rather than ‘stdout’ – this can assist in using tool that automatically process Catalina and LCC output, since stdout and stderr can be captured separately.
19. Fixed a problem with the Linux makefile for making library libci.
20. The linux sources and binaries now include srecord version 1.55, while the Windows sources and binaries include srecord version 1.47. This is because srecord version 1.47 is the only version that has a pre-built version for Windows, but this version does not compile with some recent Linux distros. There should be no differences in most cases.

Release 2.3:

21. Fixed a problem with the names of initializers for variables with file scope - previously Catalina used temporary file names to try and make sure names did not collide between files - but on Windows the names of temporary files get reused often enough to cause problems when compiling programs with more than about 20 files. This could lead to Catalina using the same name in multiple files, causing the link to fail with the same symbol defined multiple times. Now Catalina also adds the compile time to the names of the variables.
22. Fixed a bug in the _coginit() function when using the LARGE memory model.
23. Fixed a problem with temporary file names - the names generated were too long for file systems that can only accept DOS 8.3 filenames.
24. Fixed a bug in the Proxy SD driver that meant programs that tried to access the SD card before the proxy had started could fail - now the proxy drivers wait until the proxy server indicates it has completed initialization before proceeding.
25. Included a new library option - libci. This library is used by specifying '-lci' on the command line instead of '-lc'. This library omits code for input and output of floating point numbers - this can save up to 8Kb even on a simple program like 'hello_world.c'. Note that this only affects the printf and scanf functions - floating point can still be used within the program.
26. Modified the programs 'lcc', 'bind' and 'catalina' to allow for spaces in file and path names under Windows. Now the only time short path names must be used is when using gnu make to build lcc (this is due to a known limitation of GNU make, which can't cope with spaces in filenames used in 'make' rules).

27. Include file processing was not working as described in the documentation for multiple include multiple paths - now fixed.
28. Updates to the floating point plugins to include recent fixes by Cam Thompson to the exp() and pow() functions.
29. Catalina now has Code::Blocks support, which provides a graphical user environment for editing and compiling Catalina C programs. See the file README.Codeblocks in the codeblocks subdirectory for more details.

Release 2.2:

1. Added Morpheus support. XMM is supported on CPU #2. The Mem+ board is also supported, but Morpheus and the Mem+ have to be configured to provide a contiguous block of XMM RAM for Catalina to use.
2. Added support for a HiRes VGA display on Morpheus CPU #2. This is identical to the HiRes VGA display on other platforms except that on Morpheus it supports 256 colours (only 64 on other platforms). Also, note that this driver uses 3 cogs on Morpheus, whereas it requires only 2 cogs on other platforms - this may affect some programs that need to use other plugins as well.
3. Added error messages if you try to specify incompatible HMI options. For example, on Morpheus, you cannot use the TV HMI plugins, and Catalina will now generate an error messages if you try to compile programs that try to do so. Note that this is not guaranteed to detect all incompatible options.
4. Implemented the fseek options SEEK_CUR and SEEK_END.
5. Fixed a bug that caused Catalina to crash if a command line option that required a value was not followed by one (e.g. -D at the end of the line).
6. The XMM support code in all files that need it is now identical – the particular XMM functions required are now specified using #defines instead of commenting out parts of the code.
7. Added an XMM memory tester - see the 'ram_test' subdirectory. Useful for testing ports to new XMM hardware - this is a normal SPIN/PASM program which tests out XMM RAM - if this program runs ok, chances are good that Catalina XMM support will work fine.
8. Catalina can now initialize and run all 8 cogs on startup. Previous versions could only initialize and run 7 cogs on startup - the 8th cog could only be used by starting it manually from within the C program.
9. Individual CPUs are now called "CPU_1", "CPU_2", "CPU_3" (etc) on all multi-CPU systems. Previously on the TriBladeProp they were called “BLADE_1", "BLADE_2" etc, but this makes less sense on other platforms so Catalina now uses a consistent terminology across all multi-CPU systems. The CPU should now be specified on all compiles in multi-CPU systems - this will help reduce confusion, and also allow Catalina to detect and report incompatible options.
10. Added proxy drivers for all HMI devices (keyboard, mouse and screen) as well as the SD Card in multi-CPU systems. This allows programs running on one CPU to use the HMI devices or SD card attached to another CPU. A proxy

server program must be run on the CPU with the physical devices connected, and proxy drivers must be used on the client CPU. The use of these proxy drivers is documented in the Catalina Reference Manual.

11. Some of the TriBladeProp documentation (e.g. the README.TriBladeProp file in the target directory) was out of date and has been rewritten.
12. Fixed a problem in the keyboard support that could cause a program to read incorrect data from the keyboard, or crash.

Release 2.1:

1. Fixed a bug with C structures not being initialized or copied correctly.
2. Tidied up the build scripts to work correctly under Windows using only MinGW and MSYS, and updated various build instructions and notes.
3. Minor documentation corrections.
4. Added a note about Catalina symbols vs C symbols.

Release 2.0:

5. The Catalina Binder is now called **bind**, to make way for using **catalina** as the name of a new compiler front-end. This new front-end simplifies the entry of the many compiler and binder options, and also supports using environment variables instead of having to specify everything on the command line for each compile.
6. New standard XMM API to simplify adding XMM support for new platforms. The new API also defines several new primitives required to support the "Large" addressing mode. The Large addressing mode uses XMM RAM for all code, data and heap and only uses the Hub RAM for stack space (and local variables). The combined code, data and heap space can now be up to 16Mb (provided the platforms XMM hardware supports it).
7. A new PropTerminal HMI plugin and target is provided. This allows the use of the PropTerminal terminal emulator on a PC to provide full screen, keyboard and mouse support for Propeller platforms with no built-in HMI devices. A new demo program (test_propterm.c) demonstrates the use of the plugin. Note that due to the sharing of hardware pins, on the Hybrid and the Hydra the PropTerminal HMI plugin cannot be used at the same time as the HX512 SRAM card - this means it can essentially only be used by LMM programs.
8. All Catalina binaries now live in a 'bin' directory - this tidies up the directory structure.
9. Catalina utilities (such as the Generic SD Loader) and various other files now live in the 'utilities' directory.
10. Fixed some minor issues with lcc options not working correctly, including setting the LCCDIR.
11. Specifying the program memory sizes now accepts abbreviations using 'k' or 'm' (for kilobytes and megabytes) - e.g. 128k or 2m.
12. The default targets (lmm_default.spin, emm_default.spin, xmm_default.spin) now accept configuration via symbols defined on the command line (or in environment variables). These symbols allow most of the dedicated targets of

the previous releases to be replaced with just the default targets plus some command line options. For example:

```
catalina test.c -D HIRES_TV -D NO_MOUSE
```

instead of:

```
lcc test.c -thires_tv_no_mouse
```

Also, new options not previously supported have been added - e.g:

```
catalina test.c -D NTSC -D EEPROM
```

Dedicated targets can still be created, but the only special dedicated target now provided is the 'debug' target. The debug target now also accepts the same symbol definitions as the other targets, allowing many more target configurations to be debugged - e.g:

```
catalina test.c -tdebug -D HIRES_TV -D NTSC -D NO_FP
```

13. All the targets now default to HiRes TV output, since a TV output seems to be the most commmonly implemented display option for propeller platforms. For a VGA display (if the platform supports it), one of the VGA symbols can be defined (i.e. VGA, HIRES_VGA or LORES_VGA) - e.g:

```
catalina test.c -D HIRES_VGA
```

14. Fully integrated support for all target platforms. Now instead of having a different target directory for each platform, all the targets currently supported now use the same target directory, with symbols used to select the specific target. Supported targets that can be selected using this method are:

    HYDRA (XMM supported using HX512)

    HYBRID (XMM supported using HX512)

    TRIBLADEPROP (XMM supported on both Blade #1 and Blade #2)

    DEMO (note that this is untested - but it should work!)

    It is still possible to create new separate target directories for other hardware configurations. However, many new targets will be able to be supported merely by editing the existing target files and adding new conditionally compiled code and/or appropriate pin definitions.

15. Fixed a problem with the debug target not working on the TriBladeProp.

16. Support for using bstc or the Parallax Propeller Tool as a SPIN compiler for Catalina programs has been removed. Homespun is now the only supported SPIN compiler.

17. The 'build' scripts or batch files in each source directory are now uniformly called 'build_all.bat' (windows) or 'build_all' (linux).

18. Fixed a problem with converting int to float and float to int generating incorrect code in some circumstances.

19. Fixed a problem with multiply and divide generating incorrect code in some circumstances.

20. Fixed a problem with the RTC plugin not working on the TriBladeProp.

21. Changes to the image format - now the memory segment layout and segment addresses are at the beginning of the image, not the end - this makes loading a little simpler.

22. A new demo program is provided for illustrating the "Large" addressing mode - this is "Super Star Trek", in the sst subdirectory of the normal demo directory.
23. The length of C symbols supported before mangling is applied is now 65 characters - this means Catalina now conforms to ANSI requirements.
24. The LoRes VGA, Lores TV, and Hires TV HMI plugins now all have a visible cursor (previously only the HiRest VGA plugin had visible cursor support). To make room for this in the HMI plugin cog, the t_bin and t_hex routines have been replaced with C library equivalents (this makes them slightly larger and slower, but most users should notice no difference). The PC and PropTerminal HMI plugins still lack a visible cursor, since cursor support has to be implemented on the local screen to work efficiently.
25. Fixed t_color HMI call which was not working correctly.
26. t_geometry now returns cols*256 + rows, to make it consistent with the other HMI Support functions.