The Birds Project

The newlib-1.9.0 Library, Plan for Software Aspects of Certification

Ronald S. Burkey, *et al.*

Version 0.01
02/24/02

The Birds Project

Approval: ______________________________

Approval: ______________________________

Approval: ______________________________

1. Purpose of the PSAC Document

This is a standard "Plan for Software Aspects of Certification" document, corresponding to the guidelines in RTCA DO-178B. It describes the general characteristics of the system and its software, certification considerations, life cycles and life-cycle data, and scheduling of the software-development effort.

2. System Overview

2.1. Overview of the System

The newlib-1.9.0 library is intended to be a C-language library of code which has been pre-certified under DO-178B, and which is therefore available for use as "previously written software" in building airborne-software applications. It corresponds to a subset of the standard C function library.

In other words, newlib-1.9.0 is not a complete system, but can be used as a software component of an airborne system without further development or certification effort, except (for example) obtaining the signoff of a DER. Note that obtaining the signoff of a DER who has *previously* approved newlib-1.9.0 is likely to be most efficient means of doing so.

The newlib-1.9.0 library software is available for free use by anyone, under the terms terms set out in the file called "COPYING.NEWLIB" provided with the source code. Briefly, a developer may link his software to newlib, free of charge, and no particular licensing modifications need to be made to his own software. However, certain copyright notices may need to be displayed. COPYING.NEWLIB is reproduced in its entirety at the end of the PSAC.

Significant portions of the certification documentation have been taken directly from (or slightly adapted from) GNUpro documentation provided by Red Hat, Inc., and for this reason the certification documentation is licensed under terms specified by Red Hat, namely: Permission is granted to make and distribute verbatim copies of this documentation, provided the copyright notice and this permission notice are preserved on all copies. Permission is granted to copy and distribute modified versions of this documentation under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one. Permission is granted to copy and distribute translations of this documentation into another language, under the above conditions for modified versions.

The newlib-1.9.0 library is commercial off-the-shelf software (COTS) that was not explicitly written for airborne applications, and therefore is not certifiable *per se* as obtained directly from its distributer, Red Hat, Inc. The present development effort is an attempt to provide satisfactory documentation and software verification for a specific version of the newlib library, namely version 1.9.0.

Other embeddable C libraries could have been chosen for this project in place of Red Hat newlib. The choice of newlib for this project reflects the following philosophical criteria:

1) To supply a minimal C-callable API appropriate for writing simple embedded airborne applications. The API should include things such as file-system functions, string manipulation functions, timekeeping functions, and so forth.

2) To retain maximal portability from CPU type to CPU type, with a minimum porting effort.

3) To use mature software that has been deployed in thousands of prior (though non-airborne) projects, and hence for which the rate of development and new-bug discovery has reached low levels.

2.2. System Functions

2.2.1. Standard Utility Functions

2.2.1.1. Description

This is a subset of the functionality (familiar to C-language programmers) represented by the stdlib.h header file.

2.2.1.2. Hardware-Software Allocation

This functionality is provided entirely in software.

2.2.2. Character Type Macros

2.2.2.1. Description

This is a subset of the functionality (familiar to C-language programmers) represented by the ctype.h header file.

2.2.3. Input and Output

2.2.3.1. Description

This is a subset of the functionality (familiar to C-language programmers) represented by the stdio.h header file.

2.2.4. Strings and Memory

2.2.4.1. Description

This is a subset of the functionality (familiar to C-language programmers) represented by the string.h header file.

2.2.5. Signal Handling

2.2.5.1. Description

This is a subset of the functionality (familiar to C-language programmers) represented by the signal.h header file.

2.2.6. Time Functions

2.2.6.1. Description

This is a subset of the functionality (familiar to C-language programmers) represented by the time.h header file.

2.2.7. Locale

2.2.7.1. Description

This is a subset of the functionality (familiar to C-language programmers) represented by the locale.h header file.

2.2.8. Reentrancy

2.2.8.1. Description

The library functions ensure that, whenever possible, there is reentrancy. However, there are some functions that can not be trivially made reentrant. Hooks have been provided to allow these functions to be used in a fully reentrant fashion. These hooks use the structure, _reent, defined in reent.h.

All functions which must manipulate global information are available in two versions.

The first version has the usual name, using a single global instance of the reentrancy structure.

The second has a different name, normally formed by prepending _ and appending _r and taking a pointer to the particular reentrancy structure to use.

2.2.9. System Calls

2.2.9.1. Description

In case the operating system (if any) provides functionality not directly provided by newlib-1.9.0, the library provides means of making a system call to the operating system.

2.2.10. Variable Argument Lists

2.2.10.1. Description

This is a subset of the functionality (familiar to C-language programmers) represented by the stdarg.h and varargs.h header files.

2.2.11. Miscellaneous Macros and Function Calls

2.2.11.1. Description

Additional functionality not present in the standard C library may also be provided.

2.3. System Architecture

Because the newlib-1.9.0 library is intended to be as portable as possible, it requires no specific system architecture.

For example, no specific requirements on the quantity or address range of RAM is made.

Furthermore, the newlib-1.9.0 library has no specific dependence upon, or knowledge of, specific hardware peripherals that may or may not be present within the system. All low-level access to such functionality occurs through driver functions provided outside of newlib-1.9.0 itself.

2.4. Processors

The newlib-1.9.0 library as distributed by Red Hat, Inc., supports a wide variety of CPU types and operating systems. However, the initial airborne certification effort for newlib-1.9.0 supports only the following CPU/operating-system combinations:

Intel ’x86 CPU family without operating system.

PowerPC without operating system.

Motorola Coldfire without operating system.

Motorola Coldfire with RTEMS operating system.

Hopefully, later versions of newlib-1.9.0 will contain other CPU/operating-system combinations.

By "without operating system", we mean that use of those functions in the newlib-1.9.0 library dependent on the presence of an operating system is disallowed. There may still be an operating system of some kind present, but newlib-1.9.0 interaction with it is not supported.

2.5. Hardware-Software Interfaces

As may be deduced from the descriptions of the hardware-software allocations in the ’System Overview’ section, newlib-1.9.0 as such has no dependence on or knowledge of the hardware.

Instead, there is a hardware-abstraction layer (HAL), whose functions are defined in the newlib-1.9.0 design data, but which are actually provided by separately (if used). These are separate from newlib-1.9.0, and have life cycles separate from newlib-1.9.0.

2.6. Safety Features

Because newlib-1.9.0 is a reusable library rather than a complete system, it does not attempt to provide safety features as such.

3. Software Overview

3.1. Resource Sharing

All functions in the newlib-1.9.0 are reentrant (except as specified in their specific descriptions, in which case reentrant alternatives are generally provided), and hence are suitable for use in conjunction with multi-tasking operating systems such as RTEMS.

Since newlib-1.9.0 is a general-purpose, portable, reusable library, no specific statements can be made about its memory requirements or CPU time requirements.

3.2. Redundancy

Not relevant, since newlib-1.9.0 is a reusable library rather than a complete system.

3.3. Multiple-Version Dissimilar Software

Not used by newlib-1.9.0.

3.4. Fault-Tolerance, Failure Detection, and Safety Monitoring

Not provided by the newlib-1.9.0 library.

3.5. Software Timing and Scheduling Strategies

Since newlib-1.9.0 is a general-purpose portable library without knowledge of underlying hardware, it does not in itself impose any requirements on software timing nor scheduling. However, certain newlib-1.9.0 functionality does indirectly relate to timing requirements. For example, the `time` function, which returns the current time, depends indirectly on the existence of a hardware real-time clock or the existence of an accurate software timer (presumably implemented via an interrupt-service routine). However, this relationship between newlib-1.9.0 and the actual timing mechanism is mediated by the Hardware Abstraction Layer not present within newlib-1.9.0 itself (i.e., which must be provided separately from newlib-1.9.0). Therefore, no specific requirements or other statements regarding software timing can be made.

4. Certification Considerations

4.1. Software Level and Means of Compliance

The software is suitable for certification via RTCA DO-178B at level C.

4.2. Justification of Software Level

Since newlib-1.9.0 is a reusable library, rather than a complete system, it requires no safety justification as such.

4.3. Potential Software Contributions to Failure Conditions

Because the conditions of use of newlib-1.9.0 cannot be known (it may be used in developing software for any software of level C, D, or E), there is no way to know how newlib-1.9.0 may potentially contribute to failure conditions. There is no justification for using newlib-1.9.0 in developing software at levels A or B with the present certification materials.

5. Software-Component Life Cycles

For this project, there is only one software component, namely the newlib-1.9.0 library, and hence only one life cycle.

5.1. Life Cycle of newlib-1.9.0 Library Development

5.1.1. Life-Cycle

Because the newlib-1.9.0 development effort described here represents the creation of certification materials for pre-existing COTS software, the development processes and their ordering are somewhat out of the ordinary.

From the standpoint of the present development effort, the life cycle of the newlib-1.9.0 library began with the Coding Process. *Explanation*: While equivalents to Planning Processes, Requirements Processes, or Design Processes may have been undertaken by the various individuals and organizatons which have provided the actual software, there is no accessible, satisfactory documentation for such processes. Consequently, for the purposes of this project, it must be assumed that these additional processes did not actually occur. Similarly considerations apply to Software Verification Processes.

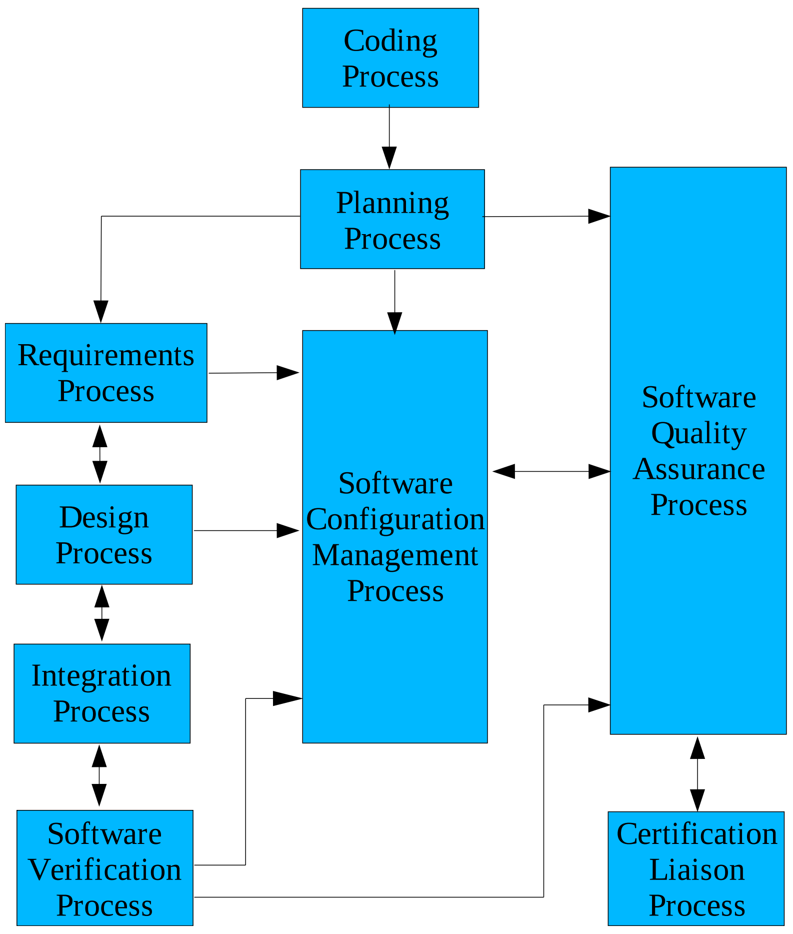
So, to reiterate: The life cycle of the newlib-1.9.0 library begins with the Coding Process. The Coding Process is followed by the Planning Process. The Planning Process, in turn, is followed by three separate chains of development processes that occur roughly simultaneously. One chain consists just of the SCM Process. Another chain consists just of the SQA Process. The third chain consists of the Requirements Process, Design Process, and Software Verification Process. For brevity, the latter will be referred to hereafter as the "RDIV chain".

Because we are merely attempting to certify pre-developed software, some of the development processes are rather abbreviated. For example, the SCM process needs merely to archive the newlib-1.9.0 library (not to provide for further development of it) and to manage the life-cycle data. The Planning, Requirements, and Design processes need merely to document work which has already been done. There is no Integration Process as such, because (newlib-1.9.0 being a reusable, hardware-independent library) there's nothing to integrate. On the other hand, the Software Verification Process is as extensive as for any other development effort.

Upon completion of the development effort, which is the release of the software by the SQA Process, life cycle data is available for input to a Certification Liaison Process. However, the Certification Liaison Process is really outside of the scope of the newlib-1.9.0 development effort, since the software produced is merely a reusable library and not a complete system.

Although not possible for the first newlib-1.9.0 release, due to non-availability of personnel, it is hoped that subsequent releases can be reviewed by DER as part of the development effort, allowing very rapid signoff of form 8110 for developers using the newlib-1.9.0 library. If this capability becomes available, it will form part of the Certification Liaison Process.

Figure 1. Life Cycle Summary


Coding Process
Planning Process
Requirements Process
Design Process
Integration Process
Software Verification Process
Software Configuration Management Process
Software Quality Assurance Process
Certification Liaison Process

5.1.1.1. Life-Cycle Processes

5.1.1.1.1. Planning Process

The Planning Process precedes all other life-cycle processes, except the Coding Process. (The Coding Process is first, since the newlib-1.9.0 development effort merely attempts to certify pre-written COTS software.) The Planning Process produces or identifies all other Plans or Standards guiding the remainder of the software-development effort. The specific aim of the newlib-1.9.0 Planning Process is to address all of the issues outlined in DO-178B section 4.0.

5.1.1.1.1.1. Transition Criteria and Satisfaction of Objectives

The Planning Process is followed by three separate chains of life-cycle processes, with the three development chains running simultaneously in parallel. These chains are the SCM Process, the SQA Process, and the RDIV chain (see the ’Life-Cycle’ section). The transitions from the Planning Process to these development chains do not necessarily occur simultaneously.

The Planning Process transitions to the RDIV chain when the PSAC, SDP, SVP, SECI, SRS, SDS, and SCS documents have all been successful reviewed and signed off.

Subsequent changes to these documents may require a return from the Design Process or Software Verification Process to the Requirements Process, but the Planning Process is never re-entered in any given life cycle.

The Planning Process transitions to the SCM Process upon successful review and signoff of the SCMP.

The Planning Process transitions to the SQA Process upon successful review and signoff of the SQAP.

5.1.1.1.2. Requirements Process

The purpose of the Requirements Process is to elucidate the existing newlib-1.9.0 software as an easy-to-grasp set of the high-level software requirements. In the case of newlib-1.9.0, this is essentially a listing of the various groups of C-language functions provided by the library. However, the intention of the newlib-1.9.0 Requirements Process is to address all of the issues outlined in DO-178B section 5.1.

The Requirements Process also addresses all considerations of DO-178B section 6.3.1, which according to DO-178B may form a part of the Software Verification Process. This can be done because at the proposed software level (’C’), there is no requirement of independence.

5.1.1.1.2.1. Transition Criteria and Satisfaction of Objectives

The Requirements Process transitions to the Design Process upon successful review and signoff of the SRD.

5.1.1.1.3. Design Process

Since the newlib-1.9.0 software is pre-existing, the main purpose of the Design Process is to detail the API definition of the various C-language functions provided by the library. However, all of the issues outlined in DO-178B section 5.2 are addressed.

The Requirements Process also addresses all considerations of DO-178B sections 6.3.2 & 6.3.3, which according to DO-178B may form a part of the Software Verification Process. This can be done because at the proposed software level (’C’), there is no requirement of independence.

5.1.1.1.3.1. Transition Criteria and Satisfaction of Objectives

The Design Process transitions to the Integration Process upon successful review and signoff of the SDD.

5.1.1.1.4. Coding Process

Since this development effort is an attempt to certify pre-existing software, there is not really a Coding Process as such. Or, more accurately, there was a Coding Process requiring a number of years to complete, but the details of this process are inaccessible to the current development effort. The current development effort, as a philosophical decision, will not modify any newlib runtime source code.

Test code not present in the Red Hat newlib v1.9.0 release, on the other hand, may be provided by this project.

5.1.1.1.4.1. Transition Criteria and Satisfaction of Objectives

The completion of the Coding Process in this development effort is a given. Real development effort begins with an essentially automatic transition to the Planning Process.

5.1.1.1.5. Integration Process

The purpose of the Integration Process is to integrate the software with the target hardware. Since the aim of the newlib-1.9.0 project is to produce a highly portable library, rather than a hardware-specific library or a physical device, there really can be no required Integration Process.

For any given target architecture, the general-purpose newlib-1.9.0 library is combined with a specific Hardware Abstraction Layer functions specific to that target. It is in the Integration Process of the hardware-abstraction layer life cycle or the application code life cycle (both of which are independent of newlib-1.9.0) that the integration occurs.

5.1.1.1.5.1. Transition Criteria and Satisfaction of Objectives

The Integration Process proceeds directly to the Software Verification Process without any effort whatever.

5.1.1.1.6. Software Verification Process

In the words of DO-178B section 6.1, "The purpose of the software verification process is to detect and report errors that may have been introduced during the software development processes." The newlib-1.9.0 Software Verification Process attempts to address all of the issues outlined in DO-178B chapter 6, except the following:

1) The Requirements Process addresses all considerations of DO-178B section 6.3.1. This can be done because at the proposed software level (’C’), there is no requirement of independence.

2) The Design Process addresses all considerations of DO-178B sections 6.3.2 & 6.3.3. This can be done because at the proposed software level (’C’), there is no requirement of independence.

3) Reviews and analysis of the outputs of the Integration Process (DO-178B section 6.3.5) are not addressed here (or elsewhere) since the entire Integration Process is optional. Refer to the ’Integration Process’ section of the PSAC for further explanation.

5.1.1.1.6.1. Transition Criteria and Satisfaction of Objectives

The Software Verification Process has several outputs:

a) Review and analysis of the source code.

b) The SVCP.

c) Review and analysis of the test results.

d) The software-test results themselves.

The Software Verification Process can transition to various other life cycle processes:

1) To the SQA Process upon successfully creating all Software Verfication Process outputs.

2) To the Coding Process upon detection of errors in software testing.

3) To the Requirements Process or Design Process upon detection of errors more appropriately resolved in the SRD or SDD than in the code.

5.1.1.1.7. Software Configuration Management Process

The Software Configuration Management Process (or just ’SCM Process’) for the most part operates simultaneously with the other life cycle processes. DO-178B chapter 7 sets out the objectives and activities of the SCM Process in some detail. In summary, the SCM Process provices the following activities:

1) Identifying configurations.

2) Implementing change control.

3) Establishing baselines.

4) Archiving the software and the life cycle data.

Though for graphical reasons the figure at the top of the ’Life-Cycle’ section of this document depicts the SCM Process as spanning merely the Requirements, Design, and Coding Processes, it actually spans all other life cycle processes, and beyond. Once data is archived by means of the SCM Process, it is theoretically intended to remain archived as long as the newlib-1.9.0 software is present within any airborne units.

Of course, since the Birds Project is not a manufacturer of airborne equipment, and the newlib-1.9.0 library is intended to be used by manufacturers of such equipment, it is not really within the capability of the Birds Project to guarantee this essentially indefinite data retention. It is therefore assumed that users of the newlib-1.9.0 library have prudently archived the version of newlib-1.9.0 they are using -- i.e., all code and life-cycle data -- within the SCM Processes of their own development efforts.

5.1.1.1.7.1. Transition Criteria and Satisfaction of Objectives

The SCM Process does not transition to other life cycle processes, since it operates in parallel with such other processes.

Ultimately, the SCM Process exists to archive life cycle data, and to perpetually maintain this archive subject to the limitations described above. The outputs of the SCM Process are the SCI and the SCM Records demonstrating archival activity. The SECI, which may legitimately be viewed as an output of the SCM Process, is actually produced by the Planning Process, but may subsequently be altered by the SCM Process.

The newlib-1.9.0 SCM Process is slightly simplified over that of a more usual airborne development effort, in that the software itself is pre-existing and is not modified within this development effort. Consequently, the only effort really required to archive and control the software itself, is to save a permanent copy of the software as received from its distributer, Red Hat, Inc.

5.1.1.1.8. Software Quality Assurance Process

The objectives and activities of the Software Quality Assurance Process (or just ’SQA Process’) are set out in chapter 8 of DO-178B. Basically, the SQA Process examines the outputs of the other life cycle processes and determines their internal consistency. In other words, it acts to assure that what has actually been accomplished is what was required to be accomplished.

Among the important activities of the SQA Process are these:

1) Establishement and management of the problem-reporting system.

2) Final release of the software.

The SQA Process operates simultaneously with the other life cycle processes, and spans the Planning Process through release of the software.

The SQA Process, at the proposed software level (’C’), is the only development process having any requirement of independence. In other words, in a usual development effort at Level C, one would expect to find the SQA function performed by independent personnel from those of the other life cycle processes.

However, the newlib-1.9.0 provides an interesting distinction from more-usual development efforts, in that the software being certified is pre-written and not modified by this development effort. Furthermore, while the Requirements Process and Design Processes do provide documentation, this documentation is not of requirements and design data produced by this development effort, but is merely a corrected form of pre-written (but non-certification) documentation created independently.

In other words, the SQA function is inherently highly independent from all actual *planning*, all *actual* design, and all *actual* coding. In light of this, there seems to be little reason to have a separate person (within the present development effort) perform the SQA functions than perform the other life-cycle processes. The newlib-1.9.0 development effort can operate as a single-individual project without compromising independence of the SQA Process in any significant way.

5.1.1.1.8.1. Transition Criteria and Satisfaction of Objectives

The SQA Process transitions to the Certification Liaison Process upon release of the software. The outputs of the SQA Process are the SQA Records.

The primary output of the SQA Process is the Software Conformity Review, which is the last step in the release of the software. The Software Conformity Review, however, takes into account not only the life cycle data in general, but also various other SQA Records.

5.1.1.1.9. Certification Liaison

Since the product of the newlib-1.9.0 development effort is a reusable library rather than an actual airborne device, there are really no certification efforts associated with it, and thus no real Certification Liaison Process.

However, since the intention of the newlib-1.9.0 project is to not merely provide software but to make it very conveniently usable by developers, a very useful certification activity would be review and approval by a DER. This "pre-approval" by a DER would allow immediate signoff of 8110 forms for developers using the newlib-1.9.0 library, very simply and relatively cheaply.

It is not known as this is written whether such DER activity will actually occur. Hence, it should be regarded as an optional activity that may be omitted.

5.1.1.1.9.1. Transition Criteria and Satisfaction of Objectives

The outputs of the Certification Liaison Process can be these:

1) Informal notification that a DER finds the life cycle data acceptable and will be available to sign off (via 8110) upon demand, for a known fee.

2) Notification of life-cycle data problems that will require repair before the DER finds the data acceptable.

Upon the former (option #1), the Certification Liaison Process ends, but there are no further processes to which a transition can occur. Of course, the availability of signoff would be published, so that newlib-1.9.0 users could be made aware of it.

Upon the latter (option #2), problem reports are filed concerning the problems which have been found. Also, a transition may (or may not) occur to an earlier life cycle process so that the problems can be fixed. The reason for this uncertainty of action is that while approval by the DER is a desirable result of the development effort, it is not a crucially necessary one from the point of view of the Birds Project. These findings are often a matter of opinion, and a different DER might view the life cycle data differently.

5.1.1.2. Organization

The Birds Project differs from most organizations involved in airborne software development, in the sense that it is not a commercial organization engaged in manufacturing a product. Instead, it is an effort to freely provide certification-friendly materials to the aviation community, as a service. Furthermore, at least at the time of inception of the newlib-1.9.0 development effort, it is not really "organized" at all and, indeed, consists of a single individual.

In contrast, the pre-written software which this development effort is attempting to certify, and the pre-written documentation which has been absorbed and corrected in this project's documentation, have been written by a large and unknown assemblage of individuals, over a period of many years. These individuals have often not been associated with any clearly definable organization. In particular, even though the pre-written newlib library, version 1.9.0, is a "product" of Red Hat, Inc., it was not written by Red Hat, Inc., and very few of the individuals involved are likely to be associated with Red Hat, Inc.

For these reasons, the organization of the newlib-1.9.0 development effort cannot be understood in terms of an "org chart" with a neatly defined flow of authority. One must instead concentrate only on the individuals involved, and even then only on the individuals involved in the certification effort as opposed to the development of the actual software.

5.1.1.3. Organizational Responsibilities

As explained earlier, because pre-written software is being certified, it is possible for the newlib-1.9.0 development effort to operate with a single individual, and still to maintain the "independence" requirements (see DO-178B Table A-9) at the proposed software level ('C').

For the initial release of newlib-1.9.0, this minimum (1 person) is used. For later releases, more or other personnel may be used. In this section, all personnel involved in all releases are identified, and their levels of involvement in these releases are made clear. Because there is no corporate entity providing authority for the development activities, we also provide brief resumes of participating individuals, so that their qualifications for their activities are clear.

Personnel for newlib-1.9.0 version 1.00

Ronald S. Burkey has a B.S. degree in Mathematics and a Ph.D. in Physics. He has been professionally involved in designing electronic hardware and firmware for airborne applications (and non-airborne applications) since 1984. He has been responsible for both DO-178A and DO-178B certification efforts.

5.1.1.4. Certification Liaison

Because newlib-1.9.0 is a reusable library rather than a complete airborne product, no direct interaction with certification authorities is required.

5.1.2. Life-Cycle Data

5.1.2.1. Data Items

Because newlib-1.9.0 is a reusable library rather than a complete airborne product, no occasion for submission of life cycle data by the Birds Project arises. Rather, all of the life cycle data items described below are made available to developers wishing to use the newlib-1.9.0 library, or to anyone else, along with newlib-1.9.0 source code. The status of this data within the development efforts of newlib-1.9.0 users cannot in principle be known to the newlib-1.9.0 development effort, and hence cannot be specified here.

The following items or categories of life cycle data items are created.

The Plan for Software Aspects of Certification (PSAC) is created by the Planning Process.

The Software Development Plan (SDP) is created by the Planning Process.

The Software Verification Plan (SVP) is created by the Planning Process.

The Software Configuration Management Plan (SCMP) is created by the Planning Process.

The Software Quality Assurance Plan (SQAP) is created by the Planning Process.

The Software Requirements Standards (SRS) are created by the Planning Process.

The Software Design Standards (SDS) are created by the Planning Process.

The Software Code Standards (SCS) are created by the Planning Process.

The Software Requirements Data (SRD) are created by the Requirements Process.

The Software Design Description (SDD) is created by the Design Process.

The Source Code is created by the Coding Process.

There is no Executable Object Code in general, since newlib-1.9.0 is a reusable library rather than a complete product. Creation of Executable Object Code is performed by integrating newlib-1.9.0 with a target-specific Hardware Abstraction Layer, which is presently visualized as a separate activity from newlib-1.9.0 development itself. For specific CPU/operating-system types that have been specified as "supported", Executable Object Code in the form of linkable libraries is provided.

The Software Verification Cases and Procedures (SVCP) and Software Verification Results (SVR) are created or completed by the Software Verification Process.

The Software Life Cycle Environment Configuration Index (SECI) is created by the Planning Process, but may be altered by the SCM Process prior to creation of the Executable Object Code (if any) or transition to the Software Verification Process.

The Software Configuration Index (SCI) is produced by the SCM Process.

Problem Reports are sanctioned and maintained by the SQA Process, but may actually be produced at any time, by anyone with access to the newlib-1.9.0 problem-reporting system.

SCM Records are produced by the SCM Process.

SQA Records are produced by the SQA Process.

The Software Accomplishment Summary (SAS) is produced by the SQA Process.

5.1.2.2. Data Relationships

All relationships among life cycle data items seem clear from the ’Life-Cycle’ section above.

5.1.2.3. Data Formats

All data items other than Source Code and Executable Object Code are made available as Adobe PDF files, viewable with the freely available Adobe Acrobat Reader program. This includes the SVR, and all SCM Records and SQA Records (such as review and audit forms). If necessary, hand-written data is scanned in order to produce the necessary PDF files.

Source Code is made available as a UNIX ’tar’ file, known as a "tarball", compressed with the ’gzip’ utility. Separate tarballs are provided for the Red Hat supplied source distribution and for the test code created by this project.

Executable Object Code (actually, linkable libraries) for the specific CPU types used in the Software Verification Process are provided as UNIX-type ’ar’ libraries.

Finally, to assure that complete file-sets for releases are available, all of the files mentioned above are combined with UNIX tar into a single tarball.

In summary, all life cycle data items are available in an on-line downloadable format rather than as hardcopies.

5.1.2.4. Means of Submitting Life-Cycle Data

Since newlib-1.9.0 is a reusable library rather than a complete product, no direct submittal of data by the Birds Project is envisaged. However, all life cycle data (code and documentation) are available to software developers, certification authorities, or any other parties, via download over the Internet.

In practice, it is envisaged that developers wishing to use newlib-1.9.0 for their projects will download the source code and other life cycle data, will archive them within their own SCM Processes, and will perform whatever submissions are required.

6. Schedule

Because newlib-1.9.0 is a library reusable by developers of airborne software, rather than for use in any specific standalone hardware-based device, there is no obvious reason for any scheduled interactions with certification authorities.

The newlib-1.9.0 library is not a commercial project with a planned release date or other milestones. Hence it can be released whenever it happens to be ready.

In summary, there is no relevant scheduling data that can be presented for the initial development effort.

7. Additional Considerations

7.1. Alternate Methods of Compliance

No qualification means alternate to DO-178B are used.

7.2. Tool Qualification

In general, a tool requires qualification if its output is used without examination. If the output of the tool is itself verified (by review, testing, or analysis) this constitutes an implicit qualification of the tool itself. With that in mind, we can examine the tools used, one-by-one, and determine their need for qualification:

Native compiler, linker, library archiver, and low-level libraries (**gcc**, **ld**, **ar**, and libgcc.a). These are used for creating desktop-computer executable code for testing purposes, or for creating embedded code if the target processor just happens to be the same as that of a common desktop computer (like Intel ’x86 or PowerPC). These tools do not require qualifiation, since their outputs (the executable code) are tested.

Cross-compiler, linker, library archiver, and low-level libraries (**gcc**, **ld**, **ar**, and libgcc.a). These are used for creating embedded Executable Object Code from a desktop computer, but for a different target CPU type. Technically, these tools also do not need qualification, since newlib-1.9.0 does not produce Executable Object Code for these environments as part of its life cycle data. Rather, separate Hardware Abstraction Layer (HAL) development efforts create this Executable Object Code. As a practical matter, however, the Birds Project wants the newlib-1.9.0 library to be as easily usable by developers as possible, and this means that this issue cannot be side-stepped. Easing HAL qualification is handled by crafting the test suites so that they can be executed in either the desktop environment or in the target environment (albeit with perhaps more effort).

Coverage tool (**gcov**). This is a tool which provides a survey of all source-code lines, indicating which of them have been exercised in testing and which have not. The coverage tool must be qualified, since there is no direct way of verifying its output. The qualification shall be done by creating a C-language program in which various bits of code are known to execute or not execute, and then to test it with **gcov**.

All other tools used either have outputs which are examined, or which feed into operations whose outputs are examined. Therefore, no other tools require qualification.

7.3. Previously-Developed Software

Additonal previously-developed software is not used by newlib-1.9.0.

7.4. Option-Selectable Software

The newlib-1.9.0 library is not, and does not contain, option-selectable software.

7.5. User-Modifiable Software

The newlib-1.9.0 library is not, and does not contain, user-modifiable software.

7.6. Commercial Off-the-Shelf Software

The newlib-1.9.0 library does not use any *additional* COTS.

7.7. Field-Loadable Software

The newlib-1.9.0 library is not field-loadable as such, but could conceivably be used by a developer producing a field-loadable program. If so, any considerations relating to this will be outlined in that developer's life cycle data.

7.8. Multiple-Version Dissimilar Software

The newlib-1.9.0 library does not use multiple-version dissimilar software.

7.9. Product Service-History

Not applicable.

8. COPYING.NEWLIB

Because the source code for newlib-1.9.0 was written by a large assemblage of developers, over a period of many years, various developers have chosen to place various licensing restrictions on the portions of code which they personally provided. In general, these provisions are very mild, and should not prevent anyone wanting to use the code from doing so, though display of certain copyright notices may be required.

Red Hat, Inc., has distilled these licensing requirements into a single document, COPYING.NEWLIB, provided with the newlib-1.9.0 source code. This document is reproduced in its entirety here, shown in a fixed-width font for emphasis: