Software Users Manual

C1XS Data Pipeline Users Manual

	Name	Signature	Date
Prepared by	Derek J. McKay		27/5/2008 21/4/2009 2/10/2009 9/12/2009 20/1/2010
Approved by			

DISTRIBUTION

Name	Iss/Rev Date	Iss/Rev Date	Iss/Rev Date	Iss/Rev Date	Iss/Rev Date	Iss/Rev Date
RAL Server	x					
C. Howe	x					
B. Maddison						
M. Grande						
B. Kellett						
K. H. Joy						
I. A . Crawford						
P. Sreekumar						
B. Swinyard						
S. Narendranath						
M. Annadurai						
C. Perry						
H. Metselaar						
D. Heather						

CHANGE RECORD

Date	Iss/Rev	Section	Comments
21/4/2009	0.2	All	Implemented bulk of the text.
2/10/2009	0.3	All	Completed manual
7/10/2009	0.4	All	Internal review
13/11/2009	0.5	All	Corrections from C.Howe
20/11/2009	0.6		
9/12/2009	0.7		Added section about data parameters
20/1/2010	0.8	4.7	Added section about file splitting

CONTENTS

1. INTRODUCTION

This document is the Software Users Manual for the C1XS Data Calibration Pipeline.

2. ACRONYMS AND ABBREVIATIONS

C1XS Chandrayaan-1 X-ray Spectrometer

XSM X-ray Solar Monitor

2.1 References

2.1.1 Applicable Documents

Along with this document the documents listed below form part of the overall software plan.

AD No	Document Number	Title
AD1	C1-C1X-RAL-RS-0017_0.5	User Requirements Definition
AD2	C1-C1X-RAL-PL-0012_0.3	Software Project Management Plan
AD3	C1-C1X-RAL-TP-0024_0.1	Test Procedure
AD4	C1-C1X-RAL-MA-0018-0.5	Software Requirements Specification

2.1.2 Reference Documents

The following documents are used for guidance and information; selected sections of the individual documents may form part of this plan and will be followed to the extent specified.

RD No	Document Number	Title
RD1	CH1-SAC-PL-001	Chandrayaan-1 Archive Plan
RD2	CH1-SAC-PL-002	Chandrayaan-1 Archive Conventions
RD3	C1-CIX-RAL-ICD-0002	C1XS/XSM Data Handling ICD
RD4	C1-C1X-RAL-TN-0014	Data Rate and volume calculations
RD5	http://ilrs.gsfc.nasa.gov/reports/ilrs_reports/9809_attach7a.html	Committee on Data Management, Archiving, and Computing (CODMAC) Data Level Definitions
RD6	C1-C1X-RAL-ICD-0008	C1XS Experimenter to Archive ICD

3. INSTALLATION

Installation of the C1XS Data Calibration Pipeline is straightforward. In order to achieve maximum compatibility across multiple UNIX-like platforms, it has been implemented as a TAR.GZ package, containing the source code and helpful build scripts. A brief summary of the installation process is included in the README file, contained in that distribution. Also included is an INSTALL file with the details of installation tools themselves. A copy of the INSTALL file is included in Section 5 of this document.

The following sections give a step-by-step description of the installation process for both UNIX/Linux and Windows (via CygWin).

3.1 Standard UNIX/Linux Installation

Once you have obtained the source code distribution, you will need to unpack it. This is done with the following command:

```
% tar -zxvf c1xs-X.Y.Z.tar.gz
```

Where X.Y.Z is the version number that appears on the distribution you are trying to install. Note that on some older systems, you may need to use the alternate “gtar” command, instead of “tar”. Once the archive has been unpacked, you will find a new directory has been created. Change into that directory with:

```
% cd c1xs-X.Y.Z
```

This is where you will find the README, INSTALL and other files. The names of these files are standard, as would be found in most typical Linux/UNIX distributions.

You can now configure your installation to your operating system. This is done using the “configure” script which you will find in this directory. The configure script requires at least one argument to indicate whether the cspice library should be used or not. The case where the cspice library is used is described below; to disable use of the cspice library, use the “--without-cspice” option. The complete list of options can be seen using “./configure --help”

By default, it will attempt to install the C1XS Data Pipeline into the system default areas (usually the /usr/local heirarchy). This is fine for users who are working on their own private machines and who have system privileges. It is also suitable for system administrators who are installing it across the entire system for all users. However, most users will probably want to install it to some local directory and this is done by using the “--prefix” option and specifying the full path of where you wish to install the software. For example:

```
% ./configure --prefix=/full/install/path
```

Please note: The installation path cannot be the same as the source code location. So, something like /home/user/c1xs is fine, but /home/user/c1xs/c1xs-0.3.0/ (namely where the ./configure script itself is) is not alright.

If SPICE kernels are being used to generate pointing information, then the path to that library is specified at this stage:

```
% ./configure --prefix=/full/install/path --with-cspice=/full/spice/path
```

SPICE kernels provide spacecraft orientation and pointing information. These kernels are read using a standard library. For this, we use the cspice version of the SPICE library.

http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/index.html
http://www-int.stsci.edu/~sontag/spicedocs/req/cspice.html

The cspice set is the library used to process the data (called kernels). You will also need the CH-1 SPICE kernels themselves. These are specific to the Chandrayaan-1 project.

Once installed, the SPICE path should have within it, the following:

```
% ls -l /home/user/cspice
total 32
drwxr-xr-x  2 user group 4096 2009-04-16 23:06 data/
drwxr-xr-x  3 user group 4096 2009-04-16 23:07 doc/
drwxr-xr-x  2 user group 4096 2009-04-16 23:06 etc/
drwxr-xr-x  2 user group 4096 2009-06-22 10:25 exe/
drwxr-xr-x  2 user group 4096 2009-04-16 23:06 include/
drwxr-xr-x  2 user group 4096 2009-06-22 10:24 lib/
-rwxr-xr--  1 user group 1773 2009-04-16 23:07 makeall.csh*
drwxr-xr-x 19 user group 4096 2009-04-16 23:07 src/
```

All paths given must be the full, absolute path starting with a slash (“/”). It is not possible to use relative directories. For the above example, the configure line would look like this:

```
% ./configure --prefix=/full/install/path --with-cspice=/home/user/cspice
```

Once the configure script has been run, the source code is now prepared for your particular system to be compiled. This is done with the standard make command.

```
% make
```

When the compilation is complete, run the test suite to ensure the pipeline is producing the expected output (note that the tests will only pass if the pipeline was built withous cspice support – see C1-C1X-RAL-TP-0024 for justification):

```
% make check
```

Then, install the software with:

```
% make install
```

When this is done, you will find two new executables,“c1xs_cal” and “c1xs_bkg_eff”, located in the “bin” directory of the path you specified with the “--prefix” option to the configure command. If you didn’t use any arguments on the configure command, they will be in the system default (typically /usr/local/bin).

If you encounter any errors during the build process, check each step again carefully. Also check the BUGS file, located in the distribution, in case there are any notes regarding your particular problem. If this doesn’t help, please contact the software developers (listed in the AUTHORS file, also located in the distribution).

Once you have built the Data Calibration Pipeline executable, you can remove the source distribution, if required.

3.2 Installing the C1XS Data Pipeline on Cygwin

These notes describe the full installation process for running the pipeline under Cygwin. Cygwin is a system that allows UNIX applications to be run on PCs that use Microsoft Windows as the operating system.

3.2.1 Installing Cygwin

1. Visit http://www.cygwin.com/
2. Download the setup file: http://www.cygwin.com/setup.exe
3. Double-click on the setup icon.
4. Step through the introduction.
5. Select "Install from Internet"
6. Select a root directory (for example C:\Cygwin). Set for all users and use Unix/Binary text file types.
7. Select a download directory
8. Select the Internet connection type
9. Select a download site (for example: http://cygwin.basemirror.de)
10. Select packages. Use the defaults, but add the following (from the "Devel" section)
    - automake (1.12 known to work, but latest version should also work)
    - gcc-core
    - make

- tcsh (from the “Shells” section, required if compiling the cspice libraries)

Some of these will auto-request other packages too, that's fine.

11. If there are any unmet dependencies, install them (use the check box at the bottom of the window to meet dependencies, DEFAULT).
12. Wait while the software downloads and builds.

3.2.2 Installing the C1XS pipeline

1. Create a local c1xs folder. In these instructions we assume: C:\Cygwin\c1xs
2. Obtain the tarball (C1XS-X_Y_Z.TGZ) from the C1XS level 4 dataset.
3. Open a Cygwin terminal (there will probably be an icon for this on your desktop).
4. Go into the c1xs directory where the TGZ file is located.
```
$  cd /c1xs
```
5. Unpack the tar archive
```
$  tar -zxvf C1XS-X_Y_Z.TGZ
```
6. Enter the c1xs directory
```
$  cd c1xs-x.y.z
```
7. Run the configuration script. Set the directory to the level above the source installation script (or somewhere at least not in the same directory). Use an absolute path.
```
$  ./configure --prefix=/c1xs --without-cspice
```
8. Build the source code:
```
$  make
```
9. Check the binaries
```
$  make check
```
10. Install the source code:
```
$  make install
```
11. If cspice support is desired, rebuild the software checking:
```
$  make distclean
$  ./configure --prefix=/c1xs --with-cspice=/PATH/TO/CSPICE
$ make
```

You can ignore the warning about it being unable to open the configuration file. If this works, then you are up and running.

3.2.3 A complete installation example

This full example assumes the user has a Cygwin installation at C:\Cygwin. It shows every step used to install the pipeline and demonstrate it works using the test data. In this

version, we have used c1xs-0.4.0.tar.gz. If you are using a later version, you will need to adjust the version number accordingly. Preliminary steps:

- Create a new folder called C:\Cygwin\c1xs
- Download c1xs-0.4.0.tar.gz to C:\Cygwin\c1xs
- Now open a Cygwin terminal and execute the following commands.

```
tar -zxvf c1xs-0.4.0.tar.gz
cd c1xs-0.4.0
./configure --prefix=/c1xs --without-cspice
make
make check
make install
```

If this all works, then you should see various log messages appear on the screen.

```
ALL: Running pipeline c1xs v0.4.0
ALL: Commencing processing
FMT: File end = "_080303_143651.LBL", Orbit = -80303, File = 143651
FMT: File end = "_080303_143651.LBL", Orbit = -80303, File = 143651
FMT: File end = "_080303_143651.LBL", Orbit = -80303, File = 143651
FMT: Loading label file "test/input/C1XS_NEHRS_080303_143651.LBL"
FMT: Input data loaded
FMT: Data is Type-12
FMT: Using detector mask 0xffffff
FMT: Loading data from "test/input/C1XS_NEHRS_080303_143651.TAB"
FMT: 448 records read from Type-12 table file
FMT: Uniform gridding applied to data
INS: Using detector mask 0xffffff
INS: Loading the spacecraft housekeeping data
INS: Loading HK label file "test/input/C1XS_NEHKD_080303_143651.LBL"
INS: Input data loaded
INS:    WARNING   Ignoring detector mask in eHkdLoad()
INS: Loading data from "test/input/C1XS_NEHKD_080303_143651.TAB"
INS: 28 records read from table file
INS: Loading the spacecraft auxiliary data
INS: Loading CZD label file "test/input/C1XS_NECZD_080303_143651.LBL"
INS: Input data loaded
INS:    WARNING   Ignoring detector mask in eCzdLoad()
INS: Loading data from "test/input/C1XS_NECZD_080303_143651.TAB"
INS: 7 records read from table file
INS: Loading the temperature-gain data
INS: Loading CGC label file "test/calib/C1XS_NACGC_20081212_151117.LBL"
INS: Input data loaded
INS:    WARNING   Ignoring detector mask in eCgcLoad()
INS: Loading CGC data from "test/calib/C1XS_NACGC_20081212_151117.TAB"
INS: 24 records read from CGC table file
INS:    WARNING   Gain correction errors are not applied
BKG: Loading the particle background data
BKG: Using detector mask 0xffffff
BKG: Loading CPB label file "test/calib/C1XS_NACPB_20081205_164046.LBL"
BKG: Input data loaded
BKG:    WARNING   Ignoring detector mask in eCpbLoad()
BKG: Loading CPB data from "test/calib/C1XS_NACPB_20081205_164046.TAB"
BKG:    WARNING   CPB integration time units are in "N/A" (assumed to be "MILLISECONDS"),
and have been converted to "SECONDS"
BKG: 501 records read from CPB table file
BKG: CPB data have been integration-time normalised by 1.600000
BKG: Applying background particle count calibration to the data
BKG:    WARNING   Calibration error interpolation is not yet provided
NRM: Loading the detector efficiency data
NRM: Using detector mask 0xffffff
NRM: Loading CDE label file "test/calib/C1XS_NACDE_20090120_161759.LBL"
NRM: Input data loaded
NRM:    WARNING   Ignoring detector mask in eCdeLoad()
NRM: Loading CDE data from "test/calib/C1XS_NACDE_20090120_161759.TAB"
NRM: 451 records read from CDE table file
NRM: Applying detector efficiency calibration to the data
NRM:    WARNING   Calibration error interpolation is not yet provided
```

```
OUT: Output file stem = "C1XS_NECCS_R00000_00000" (23 characters)
OUT: Fully-qualified output file stem = "test/output/C1XS_NECCS_R00000_00000" (35
characters)
OUT:    WARNING   SPICE kernal data are not processed yet
OUT: Writing output file "test/output/C1XS_NECCS_R00000_00000.TAB". This may take some
time...
OUT: Output TAB file written
OUT:    WARNING   Parameter "ePdsData_t *aDataPtr" not used in TXT file
OUT:    WARNING   MD5 checksums not generated
OUT: ------------------------------------------------------------------------
OUT: Detector = 20
OUT: X-range = -3439.3 to 3755.8 (range = 7195.1)
OUT: Y-range = -3.6 to 14.5 (range = 18.1)
OUT:                                    #
OUT:
OUT:
OUT:                                  #
OUT:
OUT:                                   #
OUT:                                  ###
OUT:                                  ### #
OUT:                                    #
OUT:                                   #
OUT:                                     ##
OUT:                               #      ###
OUT:                               #     # ##
OUT:                                     #
OUT:                                     ##
OUT:                               #
OUT:                                        ##  ##
OUT:                                       ######
OUT:                                #          ####  #
OUT: ###################################    ###  ####    #                     #
OUT:                                          # #### ### ##      #           ##
OUT:                                         ##  # ### ### ###### ###   ##   #
OUT:                                                ### ###################### #
OUT:                                                #     ## ##   # ###### ###
OUT: ------------------------------------------------------------------------
```

You now have a working C1XS calibration pipeline on your Cygwin installation.

4. FUNCTION

This section describes the general function of the C1XS Data Pipeline.

4.1 Binaries produced

The pipeline produces two binaries: “c1xs_cal” and “c1xs_bkg_eff”.

`c1xs_cal` reads level 2 data in PDS format (LBL + TAB) and performs offset and gain correction, and filtering of bad data. It outputs a calibrated energy scale and raw counts in PDS format (LBL + TAB) along with a text file recording the parameters used. It will also include attitude and pointing data if built with cspice support.

`c1xs_bkg_eff` reads the output of `c1xs_cal` and performs particle background and detector efficiency correction. It reads the calibration data in PDS format (LBL + TAB), but reads the data input directly from the TAB file output by `c1xs_cal`. It outputs a TAB file (without LBL) with an extra set of columns – the uncertainty in the count rate.

4.2 Parameters

The C1XS data pipeline is controlled through run-time parameters. These are parameters that can be specified when the pipeline is executed. Full descriptions can be found in Section 4.4. As of version 2.0.0, the parameters are:

Common parameters:

Parameter name	**Type**	**Description**
terminal-width	Integer	width of the terminal window
terminal-height	Integer	height of the terminal window
message-level	Level	level of output messages to terminal
log-level	Level	level of output log file records
log-file	File	name of log file for message output
config-file	File	configuration file to use
create-config	Boolean	create a new configuration file
detector-mask	Hexadecimal	hexadecimal detector mask
input-dir	Directory	Input data directory
calib-dir	Directory	calibration data directory
output-dir	Directory	output data directory
c1xs-data	File	detector input data (Type-10/11/12 label file, CCS table file)
help	Boolean	print command line parameter options, then exit
version	Boolean	display the programme version, then exit

`c1xs_cal` parameters:

integration	Integer	number of integrations
orbit-number	Integer	orbit number
file-number	Integer	file number for a given orbit
records-per-file	Integer	how many records per output file (See Section 4.8)
hk-data	File	housekeeping input data (HKD/Type-0 label file)
aux-data	File	auxiliary input data (CZD/Type-9 label file)
gain-data	File	SCD gain & offset calibration data (type-g label)
spice-dir	Directory	spice kernel directory

spice-data	File	spice meta-kernel (text file listing all kernels to load)

`c1xs_bkg_eff` parameters:

eff-data	File	SCD efficiency calibration data (type-n label)
back-data	File	background particle calibration data (label file)

The data type is one of the following:

Type	**Description**
Integer	Integer (positive or negative)
Hexadecimal	Hexadecimal number (must be prefixed by 0x)
Directory	Full or relative path to where certain files may be found
File	Full or relative file name. If the File name begins with "/" or "." it is assumed to be an absolute path or relative path respectively. Otherwise, a simple filename (e.g. file.dat) will be assumed to be found in the corresponding directory (see Section 4.3)
Boolean	Used for binary switches. Valid values can be "on" or "off"
Level	Used for error logging. (See Section 4.6.1)
Directory	Full or relative directory location. If the directory begins with "/" or "." it is assumed to be an absolute path or relative path respectively.

4.3 Parameter handling

The run-time parameters can be set in a number of different ways. Possible places that the parameters can be set are:

- Default value (hard coded)
- Configuration file (overrides the defaults)
- Command line (overrides defaults or configuration parameters)

New values will override the previous ones. When the programme starts, it will load in any default values. It will then check a configuration file and, if new values are present, will use these to override the defaults. It will then check the command line and use any values present there to override again. If multiple occurrences of a parameter occur in a configuration file or on the command line, then the last entered value will be used.

4.3.1 Command line parameters

Parameters are added on the command line using the parameter name, prefixed by a double dash (--). For example:

```
% ./c1xs_cal --help
```

or, in the case of parameters with values:

```
% ./c1xs_cal --log-level=warning
```

4.3.2 Configuration file parameters

In addition to the command line parameters, it is possible to store parameter values in a file, which can be read by the pipeline at start-up. This is described in Section 4.4.3.

4.4 Parameter descriptions

This section describes the parameters in more detail. The parameters are grouped according to functional area.

4.4.1 Terminal control

The parameters “terminal-width” and “terminal-height” are used to set tell the pipeline how big the terminal is. This is useful when running the pipeline with higher levels of log messages. If an ASCII plot is shown, then it will be displayed with the correct width and height. Most terminals default to 80 columns by 24 rows, so this is the default. These parameters have no effect of the data.

4.4.2 Message levels

The “message-level” and “log-level” parameters are used to control the verbosity of messages output by the pipeline. The “message-level” refers to terminal output and “log-level” refers to the log file.

An additional parameter “log-file” is used to specify the name of the logfile.

Message/Log levels are described in Section 4.7.

4.4.3 Configuration files

In addition to the command line parameters, it is also possible to define a configuration file, which contains the parameter values. This way, all parameters can be contained in a single file which is easier to transport and maintain. It also means that changes can be annotated and controlled by revision control systems, as the configuration file can be easily tracked in this manner.

The “config-file” parameter is used to specify the name of the configuration file used.

The “create-config” parameter can be used to create a new configuration file. This is useful for either creating a very first configuration file, which can then be edited, or as a way of saving several command line parameters for later use.

For example, the pipeline can be run with “create-config” to make the first configuration file. Then this file can be edited until the basic operations are sorted out. After running the file, it might be noticed that there is a command-line parameter set being used. The user could then simply append “--create-config” to the end of the command line to store all these parameters in an updated configuration file.

WARNING: Creating a new configuration file will automatically overwrite any previous ones. Use with care!

TIP!

When dealing with configuration files, it is often only the data directories and data file names that change from iteration to iteration. This means that identifying those parameters in a configuration file, is critical and comparing one set to another can be useful. To do this, in a consistent manner, you can use the following command set to extract *only* those parameters from the configuration file.

```
%   cat c1xs.cfg | grep -v '^#' | awk 'NF > 0' | grep '\-d' | sort
```

The ‘cat’ command lists the file, the first ‘grep’ removes the comment lines, the ‘awk’ removes any blank lines (this command is optional, but it is a good example for other command ideas) and the second ‘grep’ finds any lines with “-d” in them (the –data and –dir lines). By sorting the output, the results will always be in consistent order for comparisions.

4.5 Input files

The input files to the pipeline are PDS (Planetary Data System) format. Typically, they comprise two file components: a .LBL file and a .TAB file (label and table, respectively).

4.5.1 Cygwin and case-sensitivity

On some Cygwin systems, the filenames are not case sensitive. This can cause problems when trying to process these files on a Linux or other UNIX system. The following commands may be useful in converting everythign to upper case.

```
find . | grep '\.tab' | sed 's/\.tab//' | awk '{print "mv",$1 ".tab",$1 ".TAB"}'
find . | grep '\.lbl' | sed 's/\.lbl//' | awk '{print "mv",$1 ".lbl",$1 ".LBL"}'
```

Then, cut-and-paste the output to do the actual conversion. Although this can be done in a single step, we advocate doing it in two parts so that you can check what will be done before blindly applying it.

NOTE: Please ensure that only the appropriate files are converted. Check the output manually! These commands are great for processing a freshly unpacked tar.gz archive, but may not be suitable in a more general environment.

4.6 Output files

4.6.1 Controlling the output file name

In order to generate test files, it is often necessary to control the output filename. This is possible using the in-built file and orbit number parameters.

The C1XS Data Pipeline output has the following naming convention:

```
C1XS_NECCS_Rxxxxx_yyyyy.LBL
```

where xxxxx is the orbit number and yyyyy is the file number. For the purposes of testing the orbit number of "0" may be used. This is often the default, when using test data. The file number can then be set according to the test being performed. For example:

```
./c1xs_cal --orbit-number=0 --file-number=234
```

would generate output files:

```
C1XS_NECCS_R00000_00234.LBL
C1XS_NECCS_R00000_00234.TAB
C1XS_NECCS_R00000_00234.TXT
```

Important: Note that you must explicitly set the orbit number to zero. You cannot simply omit this parameter as the default is to revert to automatic detection, which would not give the desired result.

Also, do not use non-zero orbit numbers, except for actual science data. This is to avoid any potential confusion, in the rare case that real and test data are mixed up.

4.7 Log messages

Log messages can be reported to both a log file and the terminal output. The verbosity of these two logging streams can be controlled independently.

The parameters “log-level” and “message-level” control the message severities that can be reported to the log file and terminal respectively. The severity levels are the same as those used by the standard “syslog” system that is commonly found on most operating systems. The following table defines the different levels.

Lvl	Name	Description
-1	None	All error messages suppressed
0	Emergency	An emergency condition; that is, the system cannot be used.
1	Alert	A condition that must be corrected immediately.
2	Critical	A critical condition, such as a fatal error.
3	Error	An error message.
4	Warning	A warning message.
5	Notice	Important information, but as expected as part of standard operation.
6	Info	Verbose output, giving additional information about the operation
7	Debug	Full diagnostic output. Normally only useful for debugging.

To specify level, simply use the name (lower case). To make it easier to type, the following abbreviated versions are permitted:

none, emer, alert, crit, err, warn, notice, info, debug

Additionally, the numeric values may also be used instead of the words. In the following example, these three commands are all equivalent.

```
% ./c1xs_cal --log-level=warning
% ./c1xs_cal --log-level=warn
% ./c1xs_cal --log-level=4
```

4.8 Splitting output files

As of v1.0.1, the data pipeline provides file splitting. The reason for this is that sometimes the input files are extremely large. If the input is not split by the GDP, then it is possible to ask the data pipeline to do this instead. This will prevent extremely large files from propagating on to subsequent stages of the processing.
The parameter to control this is:

```
--records-per-file
```

which can be set to limit the number of records in an output file. So, for example, if you set --records-per-file=128 and process a file with 448 records, there will be four output files.

1 = 128 records,
2 = 128 records,
3 = 128 records,
4 = 64 records.

If --records-per-file is set to 0 or negative, then this splitting feature is disabled.

The first file in the split file sequence will have the FileNum as set either through the file name or the command line parameter. When creating subsequent files the pipeline will increment the file number by one. There is no double-checking on this. It is up to the script-level software to ascertain if this is permissible, as the pipeline is only "aware" of one file set (.LBL + .TAB) pair at a time.

Additionally, file splitting is NOT permitted on files named by date/time. An error message will be displayed to explain should it be tried accidentally.

5. APPENDIX – INSTALL FILE

The following is the installation file, as supplied in the C1XS Data Pipeline distribution. It provides details of the configure and Makefile utilities, rather than the specifics of the C1XS installation. For a worked example of how to install the C1XS Data Pipeline, please refer to Section 3.

Installation Instructions
****

Copyright (C) 1994, 1995, 1996, 1999, 2000, 2001, 2002, 2004, 2005 Free Software Foundation, Inc.

This file is free documentation; the Free Software Foundation gives unlimited permission to copy, distribute and modify it.

Basic Installation
==========

These are generic installation instructions.

The `configure' shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a `Makefile' in each directory of the package. It may also create one or more `.h' files containing system-dependent definitions. Finally, it creates a shell script `config.status' that you can run in the future to recreate the current configuration, and a file `config.log' containing compiler output (useful mainly for debugging `configure').

It can also use an optional file (typically called `config.cache' and enabled with `--cache-file=config.cache' or simply `-C') that saves the results of its tests to speed up reconfiguring. (Caching is disabled by default to prevent problems with accidental use of stale cache files.)

If you need to do unusual things to compile the package, please try to figure out how `configure' could check whether to do them, and mail diffs or instructions to the address given in the `README' so they can be considered for the next release. If you are using the cache, and at some point `config.cache' contains results you don't want to keep, you may remove or edit it.

The file `configure.ac' (or `configure.in') is used to create `configure' by a program called `autoconf'. You only need `configure.ac' if you want to change it or regenerate `configure' using a newer version of `autoconf'.

The simplest way to compile this package is:

1. `cd' to the directory containing the package's source code and type `./configure' to configure the package for your system. If you're using `csh' on an old version of System V, you might need to type `sh ./configure' instead to prevent `csh' from trying to execute `configure' itself.

   Running `configure' takes awhile. While running, it prints some messages telling which features it is checking for.

2. Type `make' to compile the package.

3. Optionally, type `make check' to run any self-tests that come with the package.

4. Type `make install' to install the programs and any data files and documentation.

5. You can remove the program binaries and object files from the source code directory by typing `make clean'. To also remove the files that `configure' created (so you can compile the package for a different kind of computer), type `make distclean'. There is also a `make maintainer-clean' target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.

Compilers and Options
==========

Some systems require unusual options for compilation or linking that the `configure' script does not know about. Run `./configure --help' for details on some of the pertinent environment variables.

You can give `configure' initial values for configuration parameters by setting variables in the command line or in the environment. Here is an example:

./configure CC=c89 CFLAGS=-O2 LIBS=-lposix

*Note Defining Variables::, for more details.

Compiling For Multiple Architectures
==========

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you must use a version of `make' that supports the `VPATH' variable, such as GNU `make'. `cd' to the directory where you want the object files and executables to go and run the `configure' script. `configure' automatically checks for the source code in the directory that `configure' is in and in `..'.

If you have to use a `make' that does not support the `VPATH' variable, you have to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use `make distclean' before reconfiguring for another architecture.

Installation Names
==========

By default, `make install' installs the package's commands under `/usr/local/bin', include files under `/usr/local/include', etc. You can specify an installation prefix other than `/usr/local' by giving `configure' the option `--prefix=PREFIX'.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you pass the option `--exec-prefix=PREFIX' to `configure', the package uses PREFIX as the prefix for installing programs and libraries. Documentation and other data files still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like `--bindir=DIR' to specify different values for particular kinds of files. Run `configure --help' for a list of the directories you can set and what kinds of files go in them.

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving `configure' the option `--program-prefix=PREFIX' or `--program-suffix=SUFFIX'.

Optional Features
==========

Some packages pay attention to `--enable-FEATURE' options to `configure', where FEATURE indicates an optional part of the package. They may also pay attention to `--with-PACKAGE' options, where PACKAGE is something like `gnu-as' or `x' (for the X Window System). The `README' should mention any `--enable-' and `--with-' options that the

package recognizes.

For packages that use the X Window System, `configure' can usually find the X include and library files automatically, but if it doesn't, you can use the `configure' options `--x-includes=DIR' and `--x-libraries=DIR' to specify their locations.

Specifying the System Type
==========================

There may be some features `configure' cannot figure out automatically, but needs to determine by the type of machine the package will run on. Usually, assuming the package is built to be run on the _same_ architectures, `configure' can figure that out, but if it prints a message saying it cannot guess the machine type, give it the `--build=TYPE' option. TYPE can either be a short name for the system type, such as `sun4', or a canonical name which has the form:

CPU-COMPANY-SYSTEM

where SYSTEM can have one of these forms:

OS KERNEL-OS

See the file `config.sub' for the possible values of each field. If `config.sub' isn't included in this package, then this package doesn't need to know the machine type.

If you are _building_ compiler tools for cross-compiling, you should use the option `--target=TYPE' to select the type of system they will produce code for.

If you want to _use_ a cross compiler, that generates code for a platform different from the build platform, you should specify the "host" platform (i.e., that on which the generated programs will eventually be run) with `--host=TYPE'.

Sharing Defaults
================

If you want to set default values for `configure' scripts to share, you can create a site shell script called `config.site' that gives default values for variables like `CC', `cache_file', and `prefix'. `configure' looks for `PREFIX/share/config.site' if it exists, then `PREFIX/etc/config.site' if it exists. Or, you can set the `CONFIG_SITE' environment variable to the location of the site script.

A warning: not all `configure' scripts look for a site script.

Defining Variables
==========

Variables not defined in a site shell script can be set in the
environment passed to `configure'. However, some packages may run
configure again during the build, and the customized values of these
variables may be lost. In order to avoid this problem, you should set
them in the `configure' command line, using `VAR=value'. For example:

```
./configure CC=/usr/local2/bin/gcc
```

causes the specified `gcc' to be used as the C compiler (unless it is
overridden in the site shell script). Here is a another example:

```
/bin/bash ./configure CONFIG_SHELL=/bin/bash
```

Here the `CONFIG_SHELL=/bin/bash' operand causes subsequent
configuration-related scripts to be executed by `/bin/bash'.

`configure' Invocation
==========

`configure' recognizes the following options to control how it operates.

`--help'
`-h'
Print a summary of the options to `configure', and exit.

`--version'
`-V'
Print the version of Autoconf used to generate the `configure'
script, and exit.

`--cache-file=FILE'
Enable the cache: use and save the results of the tests in FILE,
traditionally `config.cache'. FILE defaults to `/dev/null' to
disable caching.

`--config-cache'
`-C'
Alias for `--cache-file=config.cache'.

`--quiet'
`--silent'

`-q'
Do not print messages saying which checks are being made. To suppress all normal output, redirect it to `/dev/null' (any error messages will still be shown).

`--srcdir=DIR'
Look for the package's source code in directory DIR. Usually `configure' can determine that directory automatically.

`configure' also accepts some other, not widely useful, options. Run `configure --help' for more details.

— End of document —