- COO –

CALTECH OPTICAL OBSERVATORIES

CALIFORNIA INSTITUTE OF TECHNOLOGY

COO.WDL.TEC.001

WAVEFORM DEFINITION LANGUAGE

DOCUMENT APPROVAL

Author Release Note:

Prepared By:

_____

David Hale

Programmer

Concurrence:

_____ _____

Stephen Kaye (Add Name)

Electrical Engineer (Add Title)

_____ _____

(Add Name) (Add Name)

(Add Title) (Add Title)

Approval:

_____

(Add Name)

(Add Title)

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

No table of figures entries found.

1. INTRODUCTION

1.1 INTRODUCTION

The Waveform Definition Language (WDL) is a programming language developed by COO to generate timing scripts for the STA Archon controller, also known as Archon Configuration File, or .acf configuration (ACF) files.

1.2 PURPOSE

This document provides a reference to WDL for anyone wishing to understand how to develop a timing script (ACF file) for the STA Archon controller.

1.3 SCOPE

This subsection should

a. identify the system,
b. briefly explain what the organization of the document is.

1.4 APPLICABLE DOCUMENTS

Applicable documents contain requirements and standards that apply to the (sub-) system.

AD1–

AD2 –

1.5 REFERENCE DOCUMENTS

Reference documents are informational, and are not directly binding.

RD1– STA/Archon Manual

1.6 CHANGE RECORD

(Release Revision)	Date	Section	Modifications
(REL01)	2017 Feb 08	All	Initial draft by David Hale
	2020 Oct 20	All	Converted to COO/Sharepoint format. DH
	2020 Dec 22	All	updated. DH

1.7 ABBREVIATIONS

ACF	Archon Configuration File
ADC	Atmospheric Dispersion Compensator
AIT	Assembly, Integration and Test, conducted at the subsystem level
AIV	Assembly, Integration, and Verification, conducted at the observatory level
AO	Adaptive Optics
ARC	Astronomical Research Cameras, Inc.
CIT	California Institute of Technology
COO	Caltech Optical Observatories
DDD	Design Description Document
DRP	Data Reduction Pipeline
EE	Ensquared energy
ETC	Exposure Time Calculator
FITS	Flexible Image Transport System
FOV	Field of View
GPP	General Purpose Preprocessor
GSE	Ground Support Equipment
GUI	Graphical User Interface
ICD	Interface Control Document
ICS	Instrument Control Software
LGS	Laser Guide Star
mas	milliseconds of arc
NGS	Natural Guide Star
OCDD	Operating Concepts Description Document
OIR	(COO) Optical / Infrared Service Center
PL	Photonic Lantern
POMO	Palomar Observatory Mountain Operations
PSF	Point Spread Function
QE	Quantum Efficiency
RMS	Root Mean Square
RoN	Read out Noise
SNR	Signal to Noise Ratio
STA	Semiconductor Technology Associates, Inc.
TCS	Telescope Control System
TMT	Thirty Meter Telescope
TMTIO	TMT International Observatory
ToO	Target of Opportunity
WDL	Waveform Definition Language
WMKO	W. M. Keck Observatory

2. LANGUAGE DESCRIPTION

2.1 OVERVIEW

WDL allows the user to define waveforms and when those waveforms are active on the controller. To accomplish this, we define two types of code: waveforms and sequences. State-changing information is encoded in waveforms, while control logic is handled in sequences. Note that this is a restriction with respect to STA Archon's ACF language, where state calls and control logic can be mixed. With Archon, this restriction introduces a one clock cycle time penalty for each waveform call from a sequence.

2.2 WAVEFORMS

WDL waveforms are made up entirely of signal calls. In the context of an STA Archon, a signal is anything that can be defined in an Archon state. This includes back-plane clocks, A/D clamps, biases, etc. In WDL, each signal call is a triplet consisting of the time, signal mnemonic and signal level.

2.3 SEQUENCES

WDL sequences contain all of the control logic (aside from preprocessor directives, which allow for conditionals; see Section 4.2). Sequences can call other sequences, in order to implement nested loops, or they can call waveforms. Sequences include the following built-in Archon/ACF language elements: IF, CALL, and GOTO, parameter decrement, call iteration and RETURN.

2.4 MODES

The concept of a mode allows for assigning one or more parameters, configuration keys, and FITS header keywords into a collective group organized by a mode name. This feature requires using host software which can read and apply the mode settings. Using the appropriate host software to read this group of settings for a given mode, a command can be issued to the host to use a particular mode and by doing so, that group of settings (parameters, configuration keys, and keywords) can be applied to the Archon controller without having to reload a new timing script.

The following sections describe how WDL will treat mode sections. Note that in order to realize the benefit of modes, mode selection must be supported in the host software.

2.4.1 Definition

Modes are defined (in the .modes file) by using a mode section header of the form:

```
[MODE_xxxxx]
```

where xxxxx is any string.

2.4.2 Mode selection order

Switching between modes must not be dependent upon the order of modes selected. To understand why that is important, consider the following example of three modes, A, B, C:

```
[MODE_A]
X=100
Z=200

[MODE_B]
Y=0
Z=100

[MODE_C]
X=1
Y=1
```

If Mode B is first selected, then Mode A, the parameter values would be: X=100, Y=0, Z=200, but if mode C is first selected, then mode A, the parameter values would be X=100, Y=1, Z=200. Mode A sets X and Z but not Y, which leaves the value of Y in this example dependent on the order of the previous modes, which may produce unpredicted and unwanted behavior. To prevent this, a DEFAULT mode must be defined.

2.4.3 Default Mode

The DEFAULT mode contains at least one setting for every entry made in all possible modes to avoid the ambiguity that could be created by the order of mode selection (see Section 2.4.2).

3. FILES

WDL stores the waveform and sequence definitions in two separate files.

In addition, WDL requires a signal file which is a dictionary of the signal mnemonics used in waveform, and a module configuration (mod) file, which describes what type of controller board (what STA refers to as modules) is in each slot of the chassis, such as AD, driver, etc.. The mod file also contains all the initialization/configuration information for the controller boards.

Finally, we also require a cds file, which defines some Archon specific settings, mostly regarding the readout.

The final output of this software package is the acf file, which is the file that will ultimately be written to the Archon controller either via STA's archongui program or other means.

3.1 REQUIRED INPUT FILES

WDL uses the input files described in the following subsections. The suffixes (" .xxx ") are fixed and required as listed, but the base filename can be anything as indicated by an asterisk " * ".

3.1.1 *.conf

The .conf file is a configuration file which defines the names of the required input files. This allows one to share certain input files between projects. The contents of this file are as follows, where "*filename*" is replaced by your filename:

```
INCLUDE_FILE  = "filename.def"        /* #defines and usage are self-contained */
CDS_FILE      = "filename.cds"        /* uses #defines from .def file          */
SIGNAL_FILE   = "filename.signals"    /* #defines and usage are self-contained */
WAVEFORM_FILE = "filename.waveform"   /* uses #defines from .def and .signals  */
SEQUENCE_FILE = "filename.seq"        /* uses #defines from .def and .waveform */
MODULE_FILE   = "filename.mod"        /* #defines and usage are self-contained */
MODE_FILE     = "filename.modes"
```

3.1.2 *.def

The .def file contains user-defined macros and definitions. For example,

```
#define  _NUM_CDS_SAMPLES      20  /* The number of samples taken for CDS conversion */
#define  _FIRST_RESET_SAMPLE   40
#defeval _LAST_RESET_SAMPLE    #eval _FIRST_RESET_SAMPLE + _NUM_CDS_SAMPLES
#defeval _FIRST_VIDEO_SAMPLE   #eval _LAST_RESET_SAMPLE + 1
#define  _LAST_VIDEO_SAMPLE    _FIRST_VIDEO_SAMPLE
```

Would result in the following equivalents throughout the WDL code:

```
_NUM_CDS_SAMPLES    = 20
_FIRST_RESET_SAMPLE = 40
_LAST_RESET_SAMPLE  = 60
_FIRST_VIDEO_SAMPLE = 61
_LAST_VIDEO_SAMPLE  = 61
```

See Section 4.3 for a description of the pre-processor syntax.

3.1.3 *.cds

The .cds file contains everything that would be under the CDS/Deint tab in the archongui (Figure 1). The format of this file is as follows:

```
BIGBUF=
FRAMEMODE=
LINECOUNT=
PIXELCOUNT=
RAWENABLE=
RAWENDLINE=
RAWSAMPLES=
RAWSEL=
RAWSTARTLINE=
RAWSTARTPIXEL=
SAMPLEMODE=
SHP1=
SHP2=
SHD1=
SHD2=
TAPLINE0=
TAPLINE1=
TAPLINE2=
TAPLINE3=
TAPLINE4=
TAPLINE5=
TAPLINE6=
TAPLINE7=
TAPLINE8=
TAPLINE9=
TAPLINE10=
```

```
TAPLINE11=
TAPLINE12=
TAPLINE13=
TAPLINE14=
TAPLINE15=
TAPLINES=
TRIGOUTFORCE=
TRIGOUTINVERT=
TRIGOUTLEVEL=
TRIGOUTPOWER=
```

Note that definitions can be used here. For example, using the definitions shown in Section 3.1.2, one might have something like the following:

```
SHP1=_FIRST_RESET_SAMPLE
SHP2=_LAST_RESET_SAMPLE
SHD1=_FIRST_VIDEO_SAMPLE
SHD2=_LAST_VIDEO_SAMPLE
```

See the Archon manual for a complete description of these parameters.

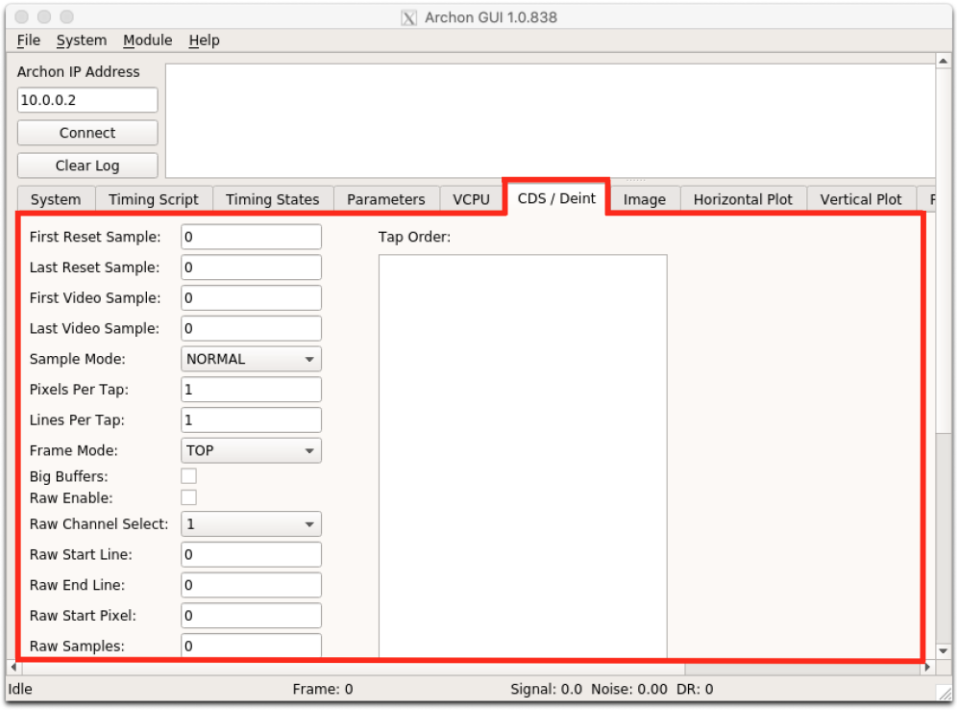

Figure 1. CDS/Deint tab from archongui

3.1.4 *.mod

The .mod file defines the modules (a.k.a. boards, or cards -- what the Archon manual refers to as "modules") installed in the Archon, the slot they are installed in, and any module-specific parameters. The syntax for the .mod file is:

`SLOT` *`n type`* { *`parameters`* }

where *`n`* is the slot number on the backplane, *`type`* is the module type, and *`parameters`* vary for each type of module. The full syntax of the SLOT command is described in Section XXX.

3.1.5 *.modes

3.1.6 *.seq

3.1.7 *.signals

3.1.8 *.waveform

3.2 TEMPORARY FILES

3.2.1 *_TMP.modules

3.2.2 *_TMP.script

3.2.3 *_TMP.states

3.2.4 *_TMP.system

3.2.5 *_TMP.wdl

3.3 OUTPUT FILE

3.3.1 *.acf

4. LANGUAGE ARCHITECTURE

4.1 DESCRIPTION

WDL is not technically a compiler but a lexer. It consists of several parts written in Python and utilizes GPP[1] for tying together the various Python scripts.

4.2 REQUIREMENTS

WDL requires Python 2.7, GPP, and GNU Make. Python 2.7 is no longer supported but its functionality can be achieved by using Anaconda and creating a Python 2.7 environment, for example:

```
conda create --name py2 python=2.7
```

Then that environment can be put into effect using:

```
conda activate py2
```

4.3 PREPROCESSOR DIRECTIVES

The ASCII waveform definition files pass through the Generic Pre-Processor (GPP) before being parsed and analyzed by the waveform compiler proper, allowing any standard GPP directive to be utilized. The most frequently used will be described below. Full documentation for GPP may be found at https://logological.org/gpp.

4.3.1 #include

`#include` *`filename`*

This causes GPP to open the specified file and evaluate its contents, inserting the resulting text in the current output. All defined user macros are still available in the included file, and reciprocally all macros defined in the included file will be available in everything that follows. The include file is looked for first in the current directory, and then, if not found, in one of the directories specified by the -I command-line option (or /usr/include if no directory was specified). Note that it is possible to put the file name between "" or <>, although these are not strictly required.

[1] Tristan Miller and Denis Auroux. GPP, the generic preprocessor. *Journal of Open Source Software*, 5(51), July 2020. ISSN 2475-9066. DOI: 10.21105/joss.02400.

4.3.2 #define

`#define` *x y*

This defines the user macro *x* as *y*. *y* can be any valid GPP input, and may for example refer to other macros. *x* must be an identifier (i.e., a sequence of alphanumeric characters and '_'). If *x* is already defined, the previous definition is overwritten. If no second argument is given, *x* will be defined as a macro that outputs nothing. Neither *x* nor *y* are evaluated; the macro definition is only evaluated when it is called, not when it is declared.

4.3.3 #defeval

`#defeval` *x y*

This acts in a similar way to *#define*, but the second argument *y* is evaluated immediately. Since user macro definitions are also evaluated each time they are called, this means that the macro *y* will undergo *two* successive evaluations. The usefulness of *#defeval* is considerable as it is the only way to evaluate something more than once, which may be needed to force evaluation of the arguments of a meta-macro that normally doesn't perform any evaluation. However since all argument references evaluated at define-time are understood as the arguments of the body in which the macro is being defined and not as the arguments of the macro itself, usually one has to use the quote character to prevent immediate evaluation of argument references.

4.3.4 #eval

`#eval` *expr*

The #eval meta-macro attempts to evaluate *expr* first by expanding macros (normal GPP evaluation) and then by performing arithmetic evaluation and/or wildcard matching. The syntax and operator precedence for arithmetic expressions are the same as in C; the only missing operators are <<, >>, ?:, and the assignment operators.

Example:

```
#define clockfreq 100000000
#define us        *(clockfreq/1000000)
#define PDELAY    #eval 10 us
```

would result in the macro `PDELAY` being defined as 100.

4.3.5 #if

`#if` *expr*

This meta-macro invokes the arithmetic/globbing evaluator in the same manner as #eval and compares the result of evaluation with the string "0" in order to begin a conditional block. In particular note that the logical value of expr is always true when it cannot be evaluated to a number.

Example:

```
#if SER_CLOCKING_DIR == FORWARD
#define SClock1 [S1TL, S1BL]
#define SClock2 [S2TL, S2BL]
#define SClock3 [S3TL, S3BL]
#else
#define SClock1 [S1TL, S1BL]
#define SClock2 [S3TL, S3BL]
#define SClock3 [S2TL, S2BL]
#endif
```

Only one group of the above #defines will be evaluated, depending on the value of SER_CLOCKING_DIR.

4.3.6 #else

`#else`

This toggles the logical value of the current conditional block. What follows is evaluated if and only if the preceding evaluation was false.

4.3.7 #elif

`#elif` *`expr`*

Read as "else if"; this meta-macro can be used to avoid nested #if conditions.

4.3.8 #endif

`#endif`

The ends a conditional block started by a #if meta-macro.

4.3.9 Comments

`/*` *`comment`* `*/`

Comments may be inserted anywhere by surrounding text with /* and */. Comments are not nestable.

5. ACF GENERATION (NEEDS WORK)

It's not really compilation but I'll call it that for now.

We need to try to document in words what Peter's powerpoint is trying to tell us. Better yet, can we convince him to write here?

5.1 FROM PETER'S POWERPOINT

5.1.1 wavgen.py converts waveforms defined by (time, board, channel, value) tuples into states and calls.

Given the timing information, wavgen determines the minimum set of machine states and the ACF script with which to call the states.

Sequences, which call waveforms or other sequences, are also parsed.

For reasons lost to time, ACF parameters are also carried along.

The output of wavgen is almost ACF-ready, with only the enumeration of tags missing.

5.1.2 Global variables and def and class definitions in wavgen

A user only needs to know how to call loadWDL().

The important global variables are UniqueStateArr and Catalog.

UniqueStateArr is an (# states) x (# channels) array, where each row is a unique state that the controller takes on.

Catalog is a python list of TimingSegment objects.

After a TimingSegment is initialized (for example, as TS) and its events are defined, a call to TS.script() or TS.plot() generates an internal call to TS.__make_states() which determines unique controller states and adds them to UniqueStateArr.

After all TimingSegments are defined, the functions state() and script() are used to generate proto-ACF texts.