LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T1300958 Advanced LIGO 5/9/2014

Front-end Control State Implementation

Stefan Ballmer, Chris Wipf, Daniel Sigg

Table of Contents

1 Introduction

The purpose of this document is to define a workflow which reduced the exposed complexity of the LIGO front-end models. It is not a programmable logic controller (PLC) or an active controls system, but tries to simplify the exposed control complexity of a front-end system by defining a number of state control variables which in turn define the state of a large number of EPICS parameters.

Looking at an auto-centering servo we typically have two quadrant detectors with four input filters each, followed by sum, pitch and yaw calculation. These values are then processed by two 2x2 input matrices, 4 servo filter bands, two 2x2 output matrix and actuation filters. This exposes a vast array of EPICS control channels to the user. Most of them have fixed values and never change, some of them may change in a scripted way and a few are adjustable.

From a user point of view most of this complexity is unwanted. The possible control states are only a few. In the simplest case, it just consists of servo on or servo off. All of the filter banks typically have fixed states, so do the input and output matrices. There may be an adjustable threshold to determine the minimum light power on a quadrant detector. This is then used to determine when to operate the auto-centering servo.

The idea proposed here is to define a small set of state control variables using a configuration file. The configuration file will list the available states for each control variable. It will further define whether a value is fixed or adjustable. If it is a fixed value, it will define it. Since we are dealing with a configuration file which can be loaded at run-time, values are not hard coded in the front-end model. Changes are no more difficult than loading a new filter. We will have the same version and configuration control as with “foton” filter files, i.e. the files will get archived whenever they get loaded to the front-end.

Channels (or tags) which are controlled by a state control variable can no longer be changed by the user directly. Using EPICS, they become effectively read-only channels. This poses a problem for commissioning, since being able to change a state on-the-fly and run a test is an important diagnostics tool. We therefore recommend that each control state has two predefined states: Default and Manual. The default state also serves as a safe state and is the default state after booting. Additional states such as Auto-center and Hold can then be added by the user.

To facilitate large front-end systems which contain multiple subtasks, we also propose to implement sub-state control variables. An ISC system may contain an auto-centering part, and ALS part and an LSC part. Each part can then define one or more sub-state control variables. Meaning, the auto-centering servo can have its own state control variable which is independent from the other parts.

The guardian control software now has a much easier task. No longer does it have to watch a large number of channels just to make sure they are in the right state. Active controls is also simplified. Instead of setting numerous control values synchronously to instigate a state change, only a few control variables have to be set. The guardian control software still has the task to recognize error conditions, deduced lock states, inform the user and put the system into the desired state.

Summarizing, this document proposes to implement front-end control states to significantly reduce the complexity of tracking, saving and referencing the interferometer status. This will provide an effective status system, implemented directly in the front-end systems whenever possible.

2 Front-end Operation

2.1 Operational Mode

- The front-end to EPICS-server channel interface will include an additional check against the configuration file. This check could be implemented on either front-end or EPICS server side. An implementation as part of the EPICS server side has the advantage that the same code could be used for controlling both front-end and Beckhoff applications.
- The desired state is commanded via additional EPICS channels specified in the configuration file (one or more to command the main state(s), additional ones to command every sub-state.
- If the commanded main model state specifies a value for a given EPICS channel, the value from the configuration file will be used in the front-end, and fed back to EPICS as a read-only variable.
- If the commanded main model state specifies a sub-state for a given EPICS channel, the value corresponding to the commanded sub-state will be used, and fed back to EPICS as a read-only variable.
- If the commanded main model state specifies manual operation for a given EPICS channel, the channel value provided by EPICS will be used.
- The configuration file can be read from disk upon request or on boot-up (like the current foton file). On boot-up the initialization values will be loaded.
- Upon loading the configuration file, a copy of the file is archived.
- The main and sub-state can be commanded through dedicated EPICS variables specified in the configuration file.
- Configuration files are under version control. When a new configuration is loaded a dated backup copy is automatically created.

2.2 Global State Machine

2.2.1 States

The program responsible for commanding the control states is itself a state machine. It has the following states:

- **Init (1):** The system is initializing. No hardware access is performed.
- **Preop (2):** The system is ready to run and access to the hardware is active. EPICS channels are manually accessible, but the controls state definitions are not enforced.
- **SafeOp (4):** The system is in a safe operating mode. Access to the hardware is active. The controls state definitions are enforced, but all tables are held in their initialization state.
- **Op (8):** The system is running normally. Access to the hardware is active. The controls state definitions are enforced, and the tables can transition to any of their predefined states.

These additional flags are available:

- **Error (16):** This indicates an error has occurred if reported by the readback channel. It is interpreted as a request to clear the error when specified in the request channel.

- **Configure (32):** This indicates that the configuration file(s) is out of date and needs to be reloaded, if reported by the readback channel. It is interpreted as a reconfiguration request when specified in the request channel.
- **Output (64):** This indicates that the outputs are disabled and that the global state machine runs in dry-mode.

The global state machine employs two controls channels:

- **(BASENAME)_STATEBITS:** This is the readback channel of the global state machine indicating its actual state. It is read-only.
- **(BASENAME)_REQUESTBITS:** This is the request channel of the global state machine which can be used to request a transition to a new state or an action. 6

2.2.2 Transitions

After start up the global state machine is set to the initialization state, while the requested state is set to restart. The restart option is a combination of mode bits and flags.

Transitions are only allowed between adjacent states. A restart request from the initialization state will first clear any errors, secondly read the configuration file and finally transition to Preop, SafeOp and Op in sequence. Similarly, if the system is in Op mode, it has to first transition through SafeOp to reach the PreOp state.

If multiple modes and flags are selected in a request, the system will first transition to the lowest indicated state. There, it will perform any actions indicated by the flags. An error will always be cleared before reading a new configuration. Finally, the system will try to transition to the highest indicated state.

If an error occurs and the system is in Op mode, it will transition back to SafeOp and set the error flag. If the error is due to a hardware access fault, the system will transition all the way back to the initialization state. With the exception of the startup a transition to a higher state always requires a user input.

2.2.3 Auxiliary Controls Channels

Bit encoded state values may not be supported well in all systems. A set of auxiliary channels can be used instead.

- **(BASENAME)_STATE:** This is a readback channel of the global state machine indicating its actual state excluding the flags and numbered 0 (INIT) through 3 (OP). It is read-only.
- **(BASENAME)_REQUEST:** This is a request channel of the global state machine excluding the flags and numbered 0 through 3.
- **(BASENAME)_ERROR_FLAG:** The error flag as a Boolean (read-only).
- **(BASENAME)_ERROR_CLEAR:** User request to clear the error.
- **(BASENAME)_CONFIG_BUSY:** Configuration is reloading (read-only).
- **(BASENAME)_CONFIG_LOAD:** User request to load a new configuration file.
- **(BASENAME)_OUTPUT_INACTIVE:** Outputs are off and inactive (read-only).
- **(BASENAME)_OUTPUT_DISABLE:** User request to disable the outputs.

3 Control File Format

3.1 General Remarks

- The file lists every EPICS channel used by the front-end model, with filter-module-related channels and matrix-related channels grouped together.
- The IFO name will be omitted in the channel list, allowing copying files between interferometers.
- For editing a dedicated GUI (like foton) will be written.
- The file will include tables to define the main state(s) and sub-states.
- In a main state table, a initialization values have to be specified. Channels not listed in any initialization list will default to zero, and will be read-only in any state other than 0 (off).
- State 0 (off) and state 1 (default) always exist. An arbitrary number of additional states can be defined.
- In the main state table, for each state and for each EPICS channel one of the following has to be specified: 1) a value, or 2) manual operation through EPICS, or 3) a pointer to the relevant sub-state table.
- An arbitrary number of sub-state tables can be provided. Unlike the main state tables, they do not have to list all channels in an initialization list.
- In the sub-state table, for each sub-state and for each involved EPICS channel one of the following has to be specified: 1) a value, or 2) manual operation through EPICS.
- For binary channels "manual" mode can be commanded for every bit individually. This is done via a manual mask.
- In addition to the default state and the user-defined states, an off or manual state is implicitly defined. It's state number is zero (0). For it, all channels default to "manual".

3.2 Example

LSC-MASTERSTATE

Channel name	INIT	0: OFF	1: DEFAULT	2: RUN
LSC-DARM_GAIN	1	manual	2	3
LSC-DARM_SW1S	0xFF	manual	0x33 (Mask: 0xF3)	0x33 (Mask: 0xF3)
LSC-CARM_GAIN	0	manual	manual	manual
LSC-MICH_GAIN	0	manual	See INIT (0)	Sub state LSC-GAINSTEPPING

LSC-GAINSTEPPING

Channel name	0: OFF	1: DEFAULT	2: STEP A	3: STEP B
LSC-MICH_GAIN	manual	See default above	1	2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--This is a configuration file of the LIGO project!-->
<ControlStateDef xmlns:p="https://dcc.ligo.org/LIGO-T1300958/public"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://dcc.ligo.org/LIGO-T1300958/public csdef.xsd">
    <Assign Name="LSC-REFL_A_RF45_I_GAIN">1.2</Assign>
    <Assign Name="LSC-REFL_A_RF45_Q_GAIN" Type="man">1.2</Assign>
    <Table Name="LSC-MASTERSTATE" Type="main" Location="internal">
        <Assign Name="LSC-DARM_GAIN">1</Assign>
        <Assign Name="LSC-DARM_SW1S">0xF3</Assign>
        <Assign Name="LSC-CARM_GAIN">0</Assign>
        <Assign Name="LSC-MICH_GAIN">0</Assign>
        <State Number="0" Name="Off" Ramp="0"/>
        <State Number="1" Name="Default">
            <Assign Name="LSC-DARM_GAIN" Type="val" >2</Assign>
            <Assign Name="LSC-DARM_SW1S" Type="val" Ramp="0"
                Mask="0xF3">0x30</Assign>
            <Assign Name="LSC-CARM_GAIN" Type="man"/>
        </State>
        <State Number="2" Name="RUN">
            <Assign Name="LSC-DARM_GAIN" Type="val" RAMP="3.0">3</Assign>
            <Assign Name="LSC-DARM_SW1S" Type="val" Ramp="0"
                Mask="0xF3">0x33</Assign>
            <Assign Name="LSC-CARM_GAIN" Type="man"/>
            <Assign Name="LSC-MICH_GAIN" Type="sub">
                <![CDATA["LSC-GAINSTEPPING"]]>
            </Assign>
        </State>
    </Table>
    <Table Name="LSC-GAINSTEPPING" Type="sub" Location="internal">
        <State Number="0" Name="Off"/>
        <State Number="1" Name="Default"/>
        <State Number="2" Name="STEP A" RAMP="1.0">
            <Assign Name="LSC-MICH_GAIN" Type="val">1</Assign>
        </State>
        <State Number="3" Name="STEP B" RAMP="1.0">
            <Assign Name="LSC-MICH_GAIN" Type="val">2</Assign>
        </State>
    </Table>
</ControlStateDef>
```

3.3 XML Representation

We are using an xml file format. In XML, several characters are part of the syntactic structure of XML and will not be interpreted as themselves, if simply placed within an XML data source. A special character sequence needs to be used instead. These special characters are:

Special Character	&	‘	“	<	>
Escape Sequence	&amp;	&apos;	&quot;	&gt;	&lt;
Purpose	Ampersand	Single quote	Double quote	Greater than	Less than

3.3.1 CDATA Tag

Sometimes escaping a string is cumbersome. For these cases XML provides the CDATA construct. Anything between the inner square brackets of “<![CDATA[]]>” is interpreted as character data. The only restriction is that data cannot contain the sequence “]]>”.

Example: <![CDATA[“LSC-GAINSTEPPING”]]> allows the use of quotes directly.

3.3.2 Header

The XML header is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This is a configuration file of the LIGO project!-->
```

3.3.3 ControlStateDef Tag

This xml tag describes the top level or root container. It is used to define a set of tables, states and assignments. It can also include rules, conditions and includes. The latter are processed during parsing and are intended to aid in complex setups.

The xml schema can be found at https://dcc.ligo.org/LIGO-T1300958/public/csdef.xsd.

Attribute xmlns: XML namespace; set to “https://dcc.ligo.org/LIGO-T1300958/public”.

Attribute xmlns:xsi: Set to “http://www.w3.org/2001/XMLSchema-instance”.

Attribute xsi:schemaLocation: Set to “https://dcc.ligo.org/LIGO-T1300958/public csdef.xsd”.

A boilerplate XML file header can be found below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This is a configuration file of the LIGO project!-->
<ControlStateDef xmlns:p="https://dcc.ligo.org/LIGO-T1300958/public"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="https://dcc.ligo.org/LIGO-T1300958/public csdef.xsd">

</ControlStateDef>
```

3.3.4 Group Tag

This xml tag groups items together. This is a purely graphical construct, the parser will ignore it. Group tags can be used inside the ControlStateDef, Table or State tag to group assignments. Group tags can be nested.

3.3.5 Table Tag

This xml tag describes a controlled state.

Attribute Name: Specifies the tag or channel name controlling the states defined in this table.

Attribute Mask: Specifies the bit mask which is applied to the EPICS channel before a state is selected. This feature can be used to mask unimportant bits from a value.

Attribute Type: The choices are “main” or “sub”. The default is “main”, if omitted. Defines whether the table specifies a main state or a sub state

Attribute Location: The choices are “internal” or “external”. For “internal” the controlling EPICS channel is created in the EPICS server, and it is not accessible by the front-end itself. For “external” the channel has to be defined in the front-end, and thus is accessible by the front-end.

For tables of type “main”:

> Every table of type “main” has at least a State 0 (off/manual) and a State 1 (default state). If not listed explicitly, the settings in any state but 0 correspond to the initialization values. During initialization the table will be set to state 1, when not defined in the global state machine. The default value can be overwritten by explicitly listing the table name in global state machine and define an Op value. The SafeOp value is also 1 by default, but again can be overwritten in the global state machine.
>
> Multiple tables of type “main” are permissible, but their channel lists as defined in the initialization section must be disjoint.
>
> If a channel is neither listed in the initialization section of a table of type “main” or at the top level, its value is set to 0 and not changeable in any state.

For tables of type “sub”:

> The default value in any state but 0 are the initialization values of the main table pointing to it.

Attribute Ramp: Defines the default ramp time in seconds for switching values within the table. If omitted, values are changed immediately. Ramp times are not used during initialization.

Attribute Comment: Used to specify a comment.

3.3.6 Table Initialization

An Assign tag located inside the Table but outside a State tag is treated as a default value. Any state which does not define its own assignment for a channel, takes its value from the default. This is only allowed for a table of type “main”. It can only contain Assign tags of type “val” or “man” (see below). If it is defined as “val”, its value is used for the default value. When it is defined as “man”, the default is defined as manual.

If no special SafeOp value is defined within the global state machine (see below), the default value also serves as the SafeOp value. A special case is an assignment of type “man” which also has an associated valid value. In this case the default value is defined as manual, whereas the SafeOp value is derived from the specified value. If the SafeOp value is indented to be manual, but the default value is of type “val”, the tag needs a separate assignment within the SafeOp state of the global state machine using the type “man”. The ramp time for SafeOp values derived from table initialization is 0. An non-zero ramp value for SafeOp values, requires a definition within the global state machine.

The initialization list must define all channels associated with a “main” table. No exception!

Any tag that is neither listed in the initialization list of a “main” table nor at the top level, is set to and kept at zero (0).

If a tag is listed with type “val”, but has no explicit value, it is assumed to be zero (0), false (bit) or an empty string depending on the type.

3.3.7 Global State Machine

The global state machine can be referenced explicitly by defining a table of type “top”. A table of type “top” has the following predefined states: State 1 (Init), State 2 (PreOp), State 4 (SafeOp) and State 8 (Op). These states cannot be overwritten or extended.

Explicitly defining the global state machine makes the associated controls channels accessible by EPICS. The table name is used to define the base name of the controls channel.

Assign statements can only be listed for state 4 (SafeOp) and state 8 (Op). These are the only states where the state definitions are enforced, and thus the assignments associated for all other modes of the global state machines are irrelevant.

Table initialization outside the state definition is also possible. It has similar meaning to normal tables. A value assignment of type “val” is treated as a constant and the value is set to a fixed value for both SafeOp and Op modes. A value assignment of type “man” with a specified value is set to a fixed value in SafeOp but left alone in Op mode. A value assignment of type “man” without a specified value is left alone and ignored in all states.

A global initialization is an assignment tag outside any table definition. It is treated exactly like a table definition inside the “top” table.

3.3.8 State Tag

This xml tag describes an individual control state.

Attribute Number: This specifies the number of a state. This corresponds to the selection value of the controlling EPICS channel (specified in the attribute Name). The number zero (0) is reserved for the off/manual state. The number one (1) is reserved for the default state. Negative numbers are not allowed. If all the state numbers are equal or below 15, an EPICS mbbo record will be used. If at least one state value is larger than 15, an EPICS longout record will be used.

Attribute Name: This specifies the name of the state. This will be used to define the EPICS value type for mbbo records. For EPICS longout records the name is ignored. Maximum length is 16.

Channels are assigned to specific values inside a state. Omitting a channel implies inheriting the settings from the initialization section.

The State 1 (default state) of a table of type "sub" is implicitly defined as being equal to the State 1 (default state) of the main table pointing to it. However this implicit definition can be overwritten by explicitly defining a state 1.

Channels in State 1 (default state) can have manual entries and mask fields. However this should be used sparsely, and only where needed. An example might be alignment sliders, whose good settings can change from day to day.

State 0 (off) is special. In this state all channels will default to manual mode unless specified otherwise.

If the table name uses a bit mask to omit neglect certain bits from the controlling channel, this bit will be removed from the value and the more significant bits shifted down. This way states are always numbered continuously starting from zero.

Attribute Ramp: Defines the default ramp time in seconds for switching values within the state. If omitted, values are changed immediately. Ramp times cannot be specified for initialization. Ramp times specified by a state override the one specified by a table.

Attribute Comment: Used to specify a comment.

3.3.9 Assign Tag

The Assign tag is used to assign values to channel names. When defined under ControlStateDef, it denotes a global constant (using type "val") or a manual value (using type "man"). A manual value is used to initialize the channel, but is then left in manual mode.

When defined inside a table, it denotes an initial and default value for a channel. When defined under a state, it defines the value associated with a state. A channel with its optional bit mask can be defined exactly once under a table. It can be redefined inside multiple state tags of the same table, or in an associated sub state table.

When defined at the top level, it cannot be redefined inside a table. When defined inside a table, it cannot be redefined in another table.

Assign tags with the same channel name but a different bit mask are treated as different entities. If a bit mask is used, it has to match when a channel entity is reassigned inside the different states of a table.

Attribute Name: This specifies the name of a channel, e.g., "LSC_DARM_GAIN" or "LSC_DARM_SW2S".

Attribute Type: The choices are either "val", "man", "sub" or "sfm". The default, if omitted, is "val". Specifies the type of assignment made to the specified channel in the specified state. The keyword "sub", implies that a sub state Table is defined which in turn will redefine the value.

Attribute Mask: This specifies a binary mask (up to 32 bits) for the channel. If a mask is used, it has to be repeated in all associated assignments. A channel can be split into multiple entities which have non-overlapping bit masks. Multiple entities of the same channel are for all practical purposes treated like separate channels. Bits of a channel which are defined nowhere, are set and kept at zero. Omitting the Mask attribute implies that no mask is applied, i.e. all bits of the data field are used. With this definition a mask of 0 (no mask) and 0xFFFFFFFF (all bits used) are identical in purpose. A mask can be defined as a decimal, octal or hexadecimal number. The allowed number formats for mask bits are decimal, hexadecimal, octal and binary.

Attribute Sfm: The choices are "true", "1", "false" or "0". It indicates a standard filter module.

The data field of an Assign tag contains the following:

For the type "val" any of the following:

- Floating point number (e.g. 58.1, or 58E0)
- Decimal number (e.g. 58)
- Hexadecimal number (e.g 0x3A)
- Octal number (e.g. 072)
- Binary numbers (e.g. 0b00111010)
- Boolean value: true, T, false or F (case-insensitive)
- A text string in quotes, indicating an EPICS field enum value (e.g. "inactive" or "off") or a string value. Keep in mind that the angle brackets and ampersand need to be escaped; or one has to use a CDATA container.
- A command string in single quotes; see next section.

Omitting the data field for type "val" is equivalent to specifying 0 in the data field.

For the type "man" the data field is either used as an initialization value or ignored.

For the type "sub" the data field specifies the sub-table channel name. It must be a quoted string. This is only permitted in a table of type "main". It is not permitted in state 1 (default state). If a sub table has a non-zero mask associated with its name, the sub table channel name must add the bit mask by first appending a '~' character and then the mask in hex format without leading zeroes (and no "0x" prefix).

Attribute Ramp: Defines the ramp time in seconds for changing to the assigned value. Ramp times only makes sense for value assignments. The value is ramped linearly from the previous one to the newly specified one, when the state is changed. Tables can specify a default ramp time which is used to switch all values used in the table. If a state specified a different ramp time, it gets precedence over the table ramp time. Finally, a ramp time specified as part of an assignment takes precedence over all others. The ramp time of a value is set at the time the assignment is encountered. This is an important distinction, if the assignment is redefined later.

Attribute Comment: Used to specify a comment.

3.3.10 Command Strings

Command strings are text commands representing decimal values. They are best used to add a descriptive layer to a bit encoded value. Instead of specifying a binary value using 0b0101 one could instead use the form 'bits b0 b2 on'. This is particular useful for standard filer modules where one the command string 'switch io fm1 on' is a more readable than 0x06000014. Command strings are case-insensitive.

The currently supported command strings are of the form "keyword [[bit]+ cmd]+". The keyword is either "bits" or "switch" supporting a generic bit encoded value or a standard filter module. The allowed fields for 'cmd' can be one of the following:

- ON: indicates that the switch or filter stage is on,
- OFF: indicates that the switch or filter stage is off,
- MAN: indicates that the switch or filter is in manual mode, and
- "channel name"

For the type "bits" the allowed values for bit are B0 to B31 and ALL. For the type "switch" (standard filter module) the allowed values for bit are one of the following:

- INPUT: denotes the state of the input switch,
- OFFSET: denotes the state of the offset enable,
- FM1, … FM10: denotes the state of the individual filter stages,
- LIMIT: denotes the state of the limiter,
- DECIMATION: denotes the state of the decimation filter,
- OUTPUT: denotes the state of the output switch,
- HOLD: denotes the state of the hold output switch,

- ALL: includes all of the above,
- IO: includes INPUT, DECIMATION and OUTPUT, and
- FMALL: includes all filters stages.

All other switches which are not listed are set to off. A bit mask can be used to limit the switches which are used. The bit mask uses bit 0 for INPUT, bit 1 for OFFSET, etc.

3.4 Conditional Definitions and Replacement Rules

A set of regular expression rules can be used to expand tag names. Rules can be defined at the command line are then applied to all tag names found in a CSD file. Rules can also be defined in an XML file and included through the command line. Or, they can be defined inside a CSD file where they apply to tags within. Regular expression replacement rules are getting expanded at the time the tag name is read from file.

Together with the ability to redefine an assignment, the following scenario is possible: A CSD file with site independent definitions can be read and expanded into a site specific set of definitions. A CSD file with a few site specific definitions can then be used to refine this set of definitions to take care of site specific peculiarities.

The replacement rules can also be used to customize a CSD file to specific systems. For example, identical hardware can use a common CSD file.

Replacement rules at the command line are specified using a sed style notation. It has the format /expression/replacement/flags. Expression is the regular expression search string, whereas replacement is the string to replace it with. The flags are used to indicate a global search (letter 'g'), a case-insensitive search (letter 'i'), and non-tag search (letter 'a'). The default is case-insensitive, tag names only, and only the first occurrence of expression is replaced. For example, the command line string '-rl /IFO/H1/g' will replace all occurrences if the string IFO with H1.

3.4.1 Rule Tag

A rule tag is used to define a regular expression replacement rule. A rule tag can be listed just below a ControlStateDef container. In this case the rule is treated as a global rule which applies to all subsequent containers. Rules are evaluated in reverse order of declaration. Meaning, the most recently defined rule is applied first, and the rule which was defined first is applied last.

A rule can be defined inside a Table. In this case it is local rule which only applies to the current table. A rule defined inside a State is local to current state.

The regular expression syntax is the one from C++11 using the ECMAScript format.

Attribute Name: Contains the rule name. This is an optional argument. If a rule has a name, it can be reset or redefined. Only global rules can be named. All named rules are in global scope.

Attribute Flag: Contains flags for the regular expression evaluation. By default only the first occurrence is replaced, and the search is case-sensitive. The flag 'g' can be used to specify that the replacement rule needs to be applied to all occurrences which match the expression, and the flag 'i' can be used to specify a case-insensitive regular expression. By default rules are only applied to tag or channel names. By specifying the flag 'a' it can also be applied to include file names and conditions. By using a capital 'o' one can apply the rule to include file names and conditions only.

Attribute Comment: Used to specify a comment.

A rule typically contains one expression tag and one replacement tag. A named rule can also be empty. In this case it reset the previously defined named rule.

3.4.1.1 Expression Tag

The expression tag is used to capture a regular expression. It has no attributes. The CDATA construct can be used to avoid escaping the ampersand sign, the single quote, the double quote, the less than and the greater than.

Anything between an opening "<![CDATA[" and closing "]]>" will not be interpreted by the XML parser. (The double quotes are not part of the string.)

3.4.1.2 Replacement Tag

The replacement tag is used to capture the replacement string. It has no attributes. The string is used to replace each match. This may include a format specification and escape sequences that are replaced by the characters they represent. The CDATA construct is allowed for replacement tags.

3.4.2 Include Tag

This xml tag is used to reference an external CSD file which will be parsed as an additional input. It has to be defined within a ControlStateDef container, but not within a Table or Rule.

Attribute Name: Contains the file name. Replacement rules are evaluated for file names if they are not restricted to tags. This can be used to hand down a directory path. If a relative path is used, it is relative to the one of the current file.

Attribute Abort: Contains an error string. Aborts parsing with an error message, if include file is not found. Prints out a warning by default. If the Abort is set to "-", only a notice is printed.

Attribute Comment: Used to specify a comment.

The currently active set of global replacement rules will be passed down. The control state definitions as well as the global replacement rules found in the new file will be added to the current set. Includes can be nested but not more than 20 levels deep.

3.4.3 If, Else and ElseIf Tag

The If tag is used to instruct the parser to include the text within the container, if the specified condition is met. Otherwsie, it is ignored. The ElseIf tag must follow an If or ElseIf tag and continues the condition with another if as part of the else clause. The Else tag must follow an If or ElseIf tag and closes the condition with the else clause. The condition tags can be used anywhere below the ControlStateDef tag as long as it includes an xml tag. They can be nested too.

The If and ElseIf tag use name and match attributes. The Else tag does not.

Attribute Name: Contains a name which is compared against a match. Replacement rules are evaluated on names.

Attribute Match: Contains a match string. It can be a regular expression. Replacement rules are not evaluated on match strings.

Attribute Abort: Contains an error string. If the condition evaluates to true, the error message is displayed and the parsing process is aborted. Replacement rules are evaluated on error messages.

Attribute Comment: Used to specify a comment.

With the name set to the site location using a variable, say "$(IFO)", and with the match set to "[H|L]1", a replacement rule for "$(IFO)" can be used to select both H1 and L1 state definitions.

3.5 Tag Listing

Tag listings are the output of the xml parser. The parser will read the provided control state definition files, apply the replacement rules, decide which parts to include, and generate a listing ordered by tags (or channels). This tag is then used to enforce the state definition on a tag by tag basis. It is possible to output this list in xml format. If there is a problem with the state definition file, generating the tag list is a good diagnostics tool. Error and warning messages are printed in the process and the final product can be examined directly to resolve more subtle issues. At the moment tag lists are output only.

Do not mix up an xml tag with a channel or control tag! The first is an identifier in the xml file, whereas the latter is a control channel.

3.5.1 Channel Tag

This is an xml tag with the name ‘Tag’. It describes a controls tag or channel. A tag or channel container contains an optional list of dependents and collection of control tags.

Attribute Name: Contains the channel or tag name.

Attribute Type: Contains the type of the channel or tag. The type can be “single”, “mask” or “unknown”. The type “single” describes a channel with a single controls tag. The type “mask” describes an integer tag or channel which contains one or more control tags, each one with a different bit mask. The type “unknown” denotes an invalid channel.

3.5.2 Dependent Tag

This xml tag describes a link to a dependent channel or tag. A dependent tag needs to be re-evaluated when the originating tag changes value.

Attribute Name: Name of dependent control tag.

Attribute Mask: Bit mask of dependent control tag.

Name and mask uniquely identify a control tag. The mask can be omitted, when all bits are effected.

Dependents are listed for each tag which requires a dependent controls to be updated on value change. The two state and readback channels associated with the global state machine are listed as dependents at the top level.

3.5.3 Control Tag

This xml tag is used to describe a single control. A control can be a constant value, the lookup value or a table selection value.

Attribute Type: Contains the type of a control. The type can be “constant” for constant and manual values, or “lookup” for the lookup values.

Attribute Mask: Contains the bit mask for controls that only look at a selection of bit in an integer value.

3.5.4 Safe and Value Tags

Every control tag can contain a safe tag. Controls can also have a value tag. A safe tag describes a value which will be applied when the state machine is in SafeOp. A value tag describes a default which will be applied in Op mode, when no other value is defined. For control tags of type “constant” a value tag of type “val” defines a constant value and effectively makes the value unchangeable in Op mode, whereas a value tag of type “man” stays unrestricted.

Value tags are also used in lookup tables. In this case they must define a state attribute.

Attribute Type: Contains the type of the control. It can be “man” for a manual value or “val” for an explicitly specified value.

Attribute State: Contains the state number for value tags within a lookup tag.

Attribute Ramp: Specifies the ramp time when the value is changed.

3.5.5 Lookup Tags

The xml lookup tags are used to implement a table lookup. They are listed below a control tag ot type “lookup”. Each lookup control contains at least in lookup tag for a main table and optionally a list of lookup tags for each sub table. Lookup tags list within themselves a list of value tags. One value tag is required for each state which carries a settable value. If a state is undefined the default value of the controls is used. Value tags for states can also be of type “sub”. In this case they do not define a value but reference a sub-table instead.

Attribute Type: Contains the type of lookup. The type can be “main” for the main lookup table and “sub” for all other lookup tables.

Attribute Name: Contains the tag or channel name which is used to determine the state.

Attribute Mask: Contains the bit mask for the above tag or channel name. A bit mask is applied to the channel or tag value before the state is selected. Masked bits are removed from the channel value and the more significant bits are shifted down to support a continuous set of state numbers.

3.6 Example

The following set of example files is a basic outline for the isc front-end targets. The intent is to run the command

```
cdsinfo -ot -rl /%target%/h1iscex/o -i isc.xml -o isctags.xml
```

on the isc.xml file for each of the isc targets: asc, lsc, iscex and iscey.

isc.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--This is a configuration file of the LIGO project!-->
<ControlStateDef xmlns="https://dcc.ligo.org/LIGO-T1300958/public"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://dcc.ligo.org/LIGO-T1300958/public csdef.xsd">
    <Include Name="iscrules.xml" Abort="Need ISC rules file"/>
    <If Name="%loc%" Match="c">
        <If Name="%sys%" Match="lsc">
            <Include Name="%sys%.xml" Abort="Need LSC file"/>
            <Include Name="%ifo%%sys%.xml" Abort="-"/>
        </If>
        <ElseIf Name="%sys%" Match="asc">
            <Include Name="%sys%.xml" Abort="Need ASC file"/>
            <Include Name="%ifo%%sys%.xml" Abort="-"/>
        </ElseIf>
    </If>
    <ElseIf Name="%loc%" Match="[x|y]">
        <Include Name="iscend.xml" Abort="Need ISC file"/>
        <Include Name="%ifo%isce%loc%.xml" Abort="-"/>
    </ElseIf>
</ControlStateDef>
```

iscrules.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--This is a configuration file of the LIGO project!-->
<ControlStateDef xmlns="https://dcc.ligo.org/LIGO-T1300958/public"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://dcc.ligo.org/LIGO-T1300958/public csdef.xsd">
<!--Check valid target -->
    <If Name="%target%" Match="[h|l]1(?:lsc|asc|iscex|iscey)"/>
    <ElseIf Name="%target%" Match="%target%" Abort="Target specification missing"/>
    <Else Abort="Illegal target specification %target%"/>
<!-- Add Ifo as a prefix to channel names -->
    <Rule>
        <Expression><![CDATA[^[^:]+$]]></Expression>
        <If Name="%target%" Match="h1.*">
            <Replacement><![CDATA[H1:$&]]></Replacement>
        </If>
        <ElseIf Name="%target%" Match="l1.*">
            <Replacement><![CDATA[L1:$&]]></Replacement>
```

```
			</ElseIf>
			<Else>
				<Replacement><![CDATA[T1:$&]]></Replacement>
			</Else></Rule>
<!-- Define %ifo% -->
		<Rule Flag="o">
			<Expression><![CDATA[%ifo%]]></Expression>
			<If Name="%target%" Match="h1.*">
				<Replacement><![CDATA[h1]]></Replacement>
			</If>
			<ElseIf Name="%target%" Match="l1.*">
				<Replacement><![CDATA[h1]]></Replacement>
			</ElseIf>
			<Else>
				<Replacement><![CDATA[t1]]></Replacement>
			</Else>
		</Rule>
<!-- Define %sys% -->
		<Rule Flag="o">
			<Expression><![CDATA[%sys%]]></Expression>
			<If Name="%target%" Match="[a-z]\dasc">
				<Replacement><![CDATA[asc]]></Replacement>
			</If>
			<ElseIf Name="%target%" Match="[a-z]\dlsc">
				<Replacement><![CDATA[lsc]]></Replacement>
			</ElseIf>
			<ElseIf Name="%target%" Match="[a-z]\discex">
				<Replacement><![CDATA[iscex]]></Replacement>
			</ElseIf>
			<ElseIf Name="%target%" Match="[a-z]\discey">
				<Replacement><![CDATA[iscey]]></Replacement>
			</ElseIf>
			<Else>
				<Replacement><![CDATA[]]></Replacement>
			</Else>
		</Rule>
<!-- Define %end% -->
<!-- Replace -END_ in channel names with -X_ or -Y_ -->
		<If Name="%target%" Match="[a-z]\discex">
			<Rule Flag="o">
				<Expression>%end%</Expression>
				<Replacement>x</Replacement>
			</Rule>
			<Rule>
```

```
                    <Expression>-END_</Expression>
                    <Replacement>-X_</Replacement>
               </Rule>
          </If>
          <ElseIf Name="%target%" Match="[a-z]\discey">
               <Rule Flag="o">
                    <Expression>%end%</Expression>
                    <Replacement>y</Replacement>
               </Rule>
               <Rule>
                    <Expression>-END_</Expression>
                    <Replacement>-Y_</Replacement>
               </Rule>
          </ElseIf>
          <Else>
               <Rule Flag="o">
                    <Expression>%end%</Expression>
                    <Replacement></Replacement>
               </Rule>
          </Else>
<!-- Define %loc% -->
          <Rule Flag="o">
               <Expression>
                    <![CDATA[%loc%]]>
               </Expression>
               <If Name="%target%" Match="[a-z]\discex">
                    <Replacement><![CDATA[x]]></Replacement>
               </If>
               <ElseIf Name="%target%" Match="[a-z]\discey">
                    <Replacement><![CDATA[y]]></Replacement>
               </ElseIf>
               <ElseIf Name="%target%" Match="[a-z]\d(asc|lsc)">
                    <Replacement><![CDATA[c]]></Replacement>
               </ElseIf>
               <Else>
                    <Replacement><![CDATA[]]></Replacement>
               </Else>
          </Rule>
</ControlStateDef>
```

iscend.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--This is a configuration file of the LIGO project!-->
<ControlStateDef xmlns="https://dcc.ligo.org/LIGO-T1300958/public"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="https://dcc.ligo.org/LIGO-T1300958/public csdef.xsd">
        <Assign Name="ALS-END_WFS_A _RF_I1_GAIN">6</Assign>
        <Assign Name="ALS-END_WFS_A _RF_I2_GAIN">6</Assign>
        …
</ControlStateDef>
```

isctags.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This is a configuration file of the LIGO project!-->
<ControlStateDef xmlns="https://dcc.ligo.org/LIGO-T1300958/public"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://dcc.ligo.org/LIGO-T1300958/public cstag.xsd">
        <Tag Name="H1:ALS-X_WFS_A_RF_I1_GAIN" Type="single">
                <Control Type="constant">
                        <Safe Type="val">0B110</Safe>
                        <Value Type="val">0B110</Value>
                </Control>
        </Tag>
        <Tag Name="H1:ALS-X_WFS_A_RF_I2_GAIN" Type="single">
                <Control Type="constant">
                        <Safe Type="val">6</Safe>
                        <Value Type="val">6</Value>
                </Control>
        </Tag>
        …
</ControlStateDef>
```

3.7 Redefinition Rules

The following rules apply when parsing one or more control state definition files:

- Multiple ControlStateDef tags can be defined within a single file, or in multiple files.
- If multiple ControlStateDef tags are defined within a file, they should have a different target. If the same target name is used, the tables, states and assignments of the later definition will be added to the first.
- Generally, it is expected that each table definition has a unique name. If a later table has the same name, its states and assignments are added to the first table. It is treated as a simple reiteration.
- If a table is redefined within the same ControlStateDef tag, it should have the same type and location. If not, the later definition overwrites the previous one.
- If a table is redefined across multiple ControlStateDef tags, their location needs to be identical.
- If a table is redefined with a different type, the later definitions are ignored.
- Two state definitions are considered identical, if they are associated with the same table and if they have the same number. If a state is redefined with a different name, the later name is ignored.
- An assignment defines a tag. Two tags are considered identical, if there name and mask are the same. Tags are unique, if they either have different names, or if they have non-overlapping bit masks and identical names.
- Tags which are neither identical nor unique have the same name and overlapping bit masks. They are a mistake and result in undefined behavior.
- Tags are generally reiterated in different states of the same table. This results in different values depending on the state of system. A tag can also be reiterated in a different table, if the first table is of type “main” and the others are of type “sub”.
- If a tag is redefined within the same table and state, the later definition overwrites the previous one.
- A tag which is defined in a table of type “sub”, but is never defined in a table of type “main” is lost and will be ignored.
- However, it is a mistake to redefine a tag across different tables of type “main”. Or, to redefine a tag in a table when it has already be defined as a global constant. When conflicting tag definitions are present, all but one of the definitions will be ignored.

Generally, redefining a table or a state should be avoided. In a some cases, it may make sense to have a global control state definition file and keep site specific variations contained in separate files. Within the same file, it almost never makes sense to redefine a table or a state. Overwriting a previous tag has similar implications. Good practice is to keep the definitions of tables, states and tags within the same group to avoid confusion.

4 Command Line Parser

The program csdinfo can be used to parse a control state definition file. It can generate a new control state definition file with all rules applied and the include files and conditionals taken into account. The resulting control state definition file does not contain rules, include files and conditionals, but represents the final configuration which will be used to deduce the constraints on the control channels.

The output can also be in the tag list format. This format listed the configuration and constraints for each individual control channel or tag. This is the configuration which is used to enforce the configuration and constraints on each listed control channel or tag.

Finally, the output can be an EPICS database record file which describes the tags used by the internal tables.

The command line arguments are as follows:

```
Usage: csdinfo [opt] -i 'csdfile' -o 'outfile'
       Displays information about a control state definition file.
       -w# warning level (from 0 to 4, default is 2)
       -rl '/regex/replacement/flg' replacement rule
       -rf 'rfile' file with regular expression replacement rules
       -bzz tags will be added as zero constants and zero SafeOp values
            use m for manual, rules are applied, must precede -bf argument
       -bf 'infile' read an ASCII file for generating a tag list
       -i 'infile' input file name (stdin when omitted, empty for -)
       -ot 'tagfile' output a tag listing
       -oe 'dbfile' output EPICS db file
       -ox 'outfile' output csdef xml file (stdout when omitted)
```

Error messages are printed to the standard error output. With a warning level of 0, no messages are displayed. A level of 1 indicates error messages only, level 2 includes warnings, level 3 includes notices, and level 4 is very verbose.

The –rl argument lets the user define a rule at the command line using sed style syntax. The –rf argument lets the user specify a CSDef file with a set of rules. Rules are read in the order they are defined. Rules defined at the command line act as global rules when parsing a CSDef file.

The –bf argument specified an ASCII file with one tag name per line. Empty lines, lines which start with a # and lines which start with “RO “ are ignored. Rules are applied to the tags as they are read. Multiple –bf arguments are possible. Tags which are defined in one of these ASCII files and are not defined in the CSDef input file, will be added as constants to the output file. Using a burt request file one can make sure no channel has been left out. The –bzz, -bmz, -bmm and –bzm flags determine, if the added channels are treated as constant zeroes or manual valuesin Op and SafeOp modes.

The –i argument is used to specify an input filename. If none is specified, the standard input is taken. If – is specified as the filename, an empty CSDef file is read. The –o argument is used to specify the output filename. If none is specified, the standard output is used.

5 GUI for Editing the Configuration File

- The GUI provides an easily clickable graphical interface for setting the filter modules. (Using “ezcaswitch” logic - not “ezcawrite” logic)
- The GUI provides a formatted matrix interface of system matrices
- The GUI provides a bitmap interface of configuring binary control channel.
- For all other channels the GUI allows entering the value.
- “Manual” or “sub-state” mode has to be setable on a “per bit”, “per FMx block”, “per ON/OFF switch” or “per value” basis.
- In addition to basic edit functions, the GUI allows copying existing states.

6 Other Remarks

- On compilation the front-end code provides the list of channels and updates the existing configuration file.
- On compilation new, previously non-existing channels will be we set to a value 0 (off or disabled). In particular, active user action in the GUI will be need to define a channel as “manual”.
- Upon restart the front-end code will load the initialization values and default to the “default” state.
- Same infrastructure for Front-end and Beckhoff: Beckhoff-EPICS interface will be upgraded to be able to parse the same format of state files for Beckhoff's EPICS channels.