New features for the control system adapter.

Martin Hierholzer

2016-03-08

MT annual meeting

Motivation for adding new features.

- Currently only exchanging process variables is supported
- Potential users ask for more features before using the CSA

Motivation for adding new features.

- Currently only exchanging process variables is supported
- Potential users ask for more features before using the CSA
- The llrfCtrl server also demands for more features:
  - Consists largely of property definitions
  - Properties directly written to / read from hardware
  - Added history and limiters
  - No (or very simple) processing of the data for most variables

Motivation for adding new features.

- Currently only exchanging process variables is supported
- Potential users ask for more features before using the CSA
- The llrfCtrl server also demands for more features:
  - Consists largely of property definitions
  - Properties directly written to / read from hardware
  - Added history and limiters
  - No (or very simple) processing of the data for most variables
- Implementing those features in the control system *and* device dependent part is not feasible, would make major part of llrfCtrl server control system dependent

Types of features safe to add.

- To make the device code really CS independent, it must not rely on certain features to be present in the CS

Types of features safe to add.

- To make the device code really CS independent, it must not rely on certain features to be present in the CS
- All features added to the CSA must be either:
  - implementable for any CS, or
  - non-essential and thus optional.

Types of features safe to add.

- To make the device code really CS independent, it must not rely on certain features to be present in the CS
- All features added to the CSA must be either:
  - implementable for any CS, or
  - non-essential and thus optional.
- Ideally: default implementation for each feature based on the current CSA interface (thus CS independent)
- For non-essential features, the default implementation does nothing

Types of features safe to add.

- To make the device code really CS independent, it must not rely on certain features to be present in the CS
- All features added to the CSA must be either:
  - implementable for any CS, or
  - non-essential and thus optional.
- Ideally: default implementation for each feature based on the current CSA interface (thus CS independent)
- For non-essential features, the default implementation does nothing
- CS-specific implementations can be added for CS which support the feature directly

List of suggested features.

- Guaranteed limits on controlSystemToDevice variables
- Device-provided validators for controlSystemToDevice variables

List of suggested features.

- Guaranteed limits on controlSystemToDevice variables
- Device-provided validators for controlSystemToDevice variables
- History (non-essential optional feature)

List of suggested features.

- Guaranteed limits on controlSystemToDevice variables
- Device-provided validators for controlSystemToDevice variables
- History (non-essential optional feature)
- Name mapping for process variables (invisible to device)

List of suggested features.

- Guaranteed limits on controlSystemToDevice variables
- Device-provided validators for controlSystemToDevice variables
- History (non-essential optional feature)
- Name mapping for process variables (invisible to device)
- Engineering units for display
- Alarms
- Interrupts (e.g. llrfCtrl: synchronisation with x2timer)

List of suggested features.

- Guaranteed limits on controlSystemToDevice variables
- Device-provided validators for controlSystemToDevice variables
- History (non-essential optional feature)
- Name mapping for process variables (invisible to device)
- Engineering units for display
- Alarms
- Interrupts (e.g. llrfCtrl: synchronisation with x2timer)
- Save and restore values for groups of variables
- Conversion to selectable engineering units (e.g. optical delay stage: mm or ps)

List of suggested features.

- Guaranteed limits on controlSystemToDevice variables
- Device-provided validators for controlSystemToDevice variables
- History (non-essential optional feature)
- Name mapping for process variables (invisible to device)
- Engineering units for display
- Alarms
- Interrupts (e.g. llrfCtrl: synchronisation with x2timer)
- Save and restore values for groups of variables
- Conversion to selectable engineering units (e.g. optical delay stage: mm or ps)

Potentially problematic features:

- DAQ
- Combined data types

List of suggested features.

- Guaranteed limits on controlSystemToDevice variables
- Device-provided validators for controlSystemToDevice variables
- History (non-essential optional feature)
- Name mapping for process variables (invisible to device)
- Engineering units for display
- Alarms
- Interrupts (e.g. llrfCtrl: synchronisation with x2timer)
- Save and restore values for groups of variables
- Conversion to selectable engineering units (e.g. optical delay stage: mm or ps)

Potentially problematic features:

- DAQ
- Combined data types

Additional features implemented in extra tools/packages:

- Directly expose group of hardware registers as PV
- Automatic panel generation

Name mapping for process variables.

- Process variables are named by the device
- System integrator might want to rename variables to match another naming convention
- Also need to adapt for the CS requirements for variable names

Plugin mechanism for extra features.

- Developers should be able to add new features fitting this scheme without modifying the CSA package

Plugin mechanism for extra features.

- Developers should be able to add new features fitting this scheme without modifying the CSA package
- Suggestion: Put all new features into plugins, to have a coherent interface

Plugin mechanism for extra features.

- Developers should be able to add new features fitting this scheme without modifying the CSA package
- Suggestion: Put all new features into plugins, to have a coherent interface
- Plugins can be loaded for each ProcessVariable
- Each plugin can provide its own configuration interface exposed to the device code

Plugin mechanism for extra features.

- Developers should be able to add new features fitting this scheme without modifying the CSA package
- Suggestion: Put all new features into plugins, to have a coherent interface
- Plugins can be loaded for each ProcessVariable
- Each plugin can provide its own configuration interface exposed to the device code
- Default implementation should always just use existing CSA features

Suggested default implementations: limits and validators.

Limits:

- Minimum and maximum can be enforced by the CSA, changed value is exposed to the operator through a second control system variable

Suggested default implementations: limits and validators.

Limits:

- Minimum and maximum can be enforced by the CSA, changed value is exposed to the operator through a second control system variable

Validators:

- Functor objects provided by the device (but run by the CSA) returning a boolean whether the value is valid or not

Suggested default implementations: limits and validators.

Limits:

- Minimum and maximum can be enforced by the CSA, changed value is exposed to the operator through a second control system variable

Validators:

- Functor objects provided by the device (but run by the CSA) returning a boolean whether the value is valid or not
- Never changes the value
- Must be evaluated after applying the limits (or any other feature which might change values)

Suggested default implementations: limits and validators.

Limits:

- Minimum and maximum can be enforced by the CSA, changed value is exposed to the operator through a second control system variable

Validators:

- Functor objects provided by the device (but run by the CSA) returning a boolean whether the value is valid or not
- Never changes the value
- Must be evaluated after applying the limits (or any other feature which might change values)
- Validity of the value is exposed through a second control system variable

Suggested default implementations: limits and validators.

Limits:

- Minimum and maximum can be enforced by the CSA, changed value is exposed to the operator through a second control system variable

Validators:

- Functor objects provided by the device (but run by the CSA) returning a boolean whether the value is valid or not
- Never changes the value
- Must be evaluated after applying the limits (or any other feature which might change values)
- Validity of the value is exposed through a second control system variable
- Group validator has a list of validators and applies them all (logical AND)

Suggested default implementations: limits and validators.

Limits:

- Minimum and maximum can be enforced by the CSA, changed value is exposed to the operator through a second control system variable

Validators:

- Functor objects provided by the device (but run by the CSA) returning a boolean whether the value is valid or not
- Never changes the value
- Must be evaluated after applying the limits (or any other feature which might change values)
- Validity of the value is exposed through a second control system variable
- Group validator has a list of validators and applies them all (logical AND)

CS-specific implementations will use CS features!

Combined data types.

- Transfer variables grouped into a struct together

Combined data types.

- Transfer variables grouped into a struct together
- No guarantee for atomic transfer possible without CS support

Combined data types.

- Transfer variables grouped into a struct together
- No guarantee for atomic transfer possible without CS support
- Can we help us by adding flags when to read the group?

Expose a group of hardware registers as process variables.

- Often many PV are just passed through from/to the hardware
- Can be automated with mtca4u-deviceaccess and the CSA

Expose a group of hardware registers as process variables.

- Often many PV are just passed through from/to the hardware
- Can be automated with mtca4u-deviceaccess and the CSA
- Function to create one PV and one accessor per register in a module
- Another function to synchronise the PV and the accessor

Expose a group of hardware registers as process variables.

- Often many PV are just passed through from/to the hardware
- Can be automated with mtca4u-deviceaccess and the CSA
- Function to create one PV and one accessor per register in a module
- Another function to synchronise the PV and the accessor
- Device implementation should be able to intervene easily (e.g. to do some math before passing on the value)

Expose a group of hardware registers as process variables.

- Often many PV are just passed through from/to the hardware
- Can be automated with mtca4u-deviceaccess and the CSA
- Function to create one PV and one accessor per register in a module
- Another function to synchronise the PV and the accessor
- Device implementation should be able to intervene easily (e.g. to do some math before passing on the value)
- Should this feature go into an extra package?

List of features (for discussion).

- Guaranteed limits on controlSystemToDevice variables
- Device-provided validators for controlSystemToDevice variables
- History (non-essential optional feature)
- Name mapping for process variables (invisible to device)
- Engineering units for display
- Alarms
- Interrupts (e.g. llrfCtrl: synchronisation with x2timer)
- Save and restore values for groups of variables
- Conversion to selectable engineering units (e.g. optical delay stage: mm or ps)

Potentially problematic features:

- DAQ
- Combined data types

Additional features implemented in extra tools/packages:

- Directly expose group of hardware registers as PV
- Automatic panel generation