Channel Addressing

There is two ways of addressing channels ...

The READER Interface

The READER class provides an abstract interface to the historical and control data of the data source. The minimal READER should provide 3 methods

- *GetGroupInfo* - returns annotated list of all logging groups.
- *GetItemList* - returns list of all channels in the specified logging group.
- *GetRawData* - returns Iterator object with requested data.

The SEARCH Subsystem

ADEI has a modular search subsystem. The search capabilities are provided by the search engines which providing one-or-more search modules. Besides the search term, the search string may specify the search modules to perform search. The module parameters could be specified along with the modules. Read more...

Providing additional data using ADEI Web Interface

Examples of additional data are:

- The alarms registered during the data collection by control systems
- The system operator messages
- The information about events registered by DAQ subsystem
- Anything else...

This section describes how the additional data could be provided using ADEI Web Interface. The simplest approach is to provide a service generating an XML data, and XSLT stylesheet which would convert it to HTML. Of course it is possible to generate directly XHTML, but with XML+XSLT approach it is easier to alter presentation form and establish some automation (the XML service could be used by 3rd party client applications). After the service is ready, the new ADEI module (an extra page in the upper menu) should be created and registered in configuration.

So, the following steps should be taken to setup extra ADEI page:

- Create a service and put it in the *services/* subdirectory (see description here)
- Create a module and put it in the *modules/* directory (see description here)
- Create a XSLT stylesheet which accepts produced XML and generates HTML to be displayed and put it in the *xslt/* folder.
- Optionally, if additional parameters, like *target* for multi-task services, should be passed to the service, special update service should be created in *services/update/* directory (see details)
- The module should be registered in *$MODULES* variable in the configuration file *config.php*. Be careful, some of the global configuration options specified in the config.php located in the root folder of ADEI installation may be overriden in setups by setup-specific configuration file.

Providing Additional Control Tab

This section describes how to extend left sidebar with additional control tabs.

- As with additional pages, the new control tab should be created in the modules directory following the guidelines. It should define *module_title* variable and *moduleJS*, *modulePage* functions (the name of tab should be used instead of *module* prefix).
- The *modulePage* should provide HTML markup and *moduleJS* attach JavaScript handlers to set ADEI variables and, hence, control ADEI engine. The global variable *adei* should be used to access ADEI functionality.
- In certain cases the care on resizing should be taken. The following function registering callbacks can be useful: *adei.popup.RegisterOnCallback*, *adei.popup.RegisterOffCallback*, *adei.popup.RegisterReHeightCallback*, *adei.popup.RegisterReWidthCallback*, *control_modules.RegisterCallback*, and some others. For examples, take a look on *virtual.php* and *source.php* in the modules folder.
- Finally, the module should be enabled. Just add the control name into the $CONTROLS variable in the configuration. The *false* value in array will enforce starting of a new line of controls.

Short Introduction to JavaScript API

ADEI API:

- *adei.SetCustomProperties* will adjust ADEI variables
- *adei.OpenControl* will open specified control in the sidebar
- *adei.SwitchPopup* will open/close the specified popup
- *adei.OpenModule* will open the specified page
- *adei.SetStatus* will set an informational message in the ADEI statusbar
- *adei.ReportError* will set error message in the ADEI statusbar
- *adei.SetSuccessStatus* will set short living confirmation message in the ADEI status bar
- *adei.ProposeStatus* will set an informational message in the ADEI statusbar if nothing important is currently on display
- *adei.SetExtraStatus* will extend the current statusbar message
- *adei.!UpdateDIV* will update content of the specified DIV using supplied service
- *adei.!Search* will perform search and open results in the searchtab

Implementing context dependent controls: It is possible to track if your control is currently on display using mentioned before functions: *adei.popup.RegisterOnCallback*, *adei.popup.RegisterOffCallback*, *control_modules*.RegisterCallback. *Example is available in* source.php. *The* adei.config.GetModule() *function will return currently opened page. The page change notification callback can be requested using* adei.module.RegisterCallback *function. For better compatibility with crappy browsers like IE, it is better to alter the context of control only when it is displayed. The following JavaScript classes and functions could be useful:*

- The *SELECT* class provides useful interface to generate cascading select controls.
- Like with pages, the XML Module can be used to fill the control context.
- The simplest way to make part of control appear depending on the current page, is to define the css class for considered elements and, then, show/hide it using *cssShowClass*, *cssHideClass* functions. Unfortunately, not all HTML objects can be easily hidden in all browsers (yes, again mainly the IE is a problem). It is safer to hide div's if possible.

- The complete removal of controls is not supported in current version of ADEI. Implementation idea and/or patch are welcome ;)