Control System Studio Guide

For installers and maintainers of CSS

Kay Kasemir

Gabriele Carcassi

Control System Studio Guide: For installers and maintainers of CSS

Kay Kasemir
Gabriele Carcassi

Acknowledgements

Control System Studio (CSS) is the result of contributions from many people:

- First of all, CSS builds heavily on Eclipse.
- Matthias Clausen at the Deutsches Electronen Synchrotron started the CSS idea.
- Many people from the Canadian Light Source, Brookhaven National Laboratory, Argonne National Laboratory, ITER and other sites have contributed either through extensions, code fixes, suggestions or bug reports. Check the `@author` tags in the source code to get an idea.

The sources for this book are on SourceForge under http://cs-studio.hg.sourceforge.net/hgweb/cs-studio/docbook/.

Thanks to Gabriele Carcassi, the latest HTML version is available at http://cs-studio.sourceforge.net/docbook/, and the current PDF is at http://cs-studio.sourceforge.net/docbook/css_book.pdf.

Table of Contents

List of Figures

No Warranty

Part I. CSS Guide

This book has two parts. The first part consists of an introduction to CSS. It explains how to compile CSS from sources, how to install it, how to perform the initial setup of the archive and alarm system. It is meant to serve as a guide for those who need to install, maintain and extend CSS.

The second part of the book has reference chapters for selected plugins.

For more information on Eclipse RCP, the technology underlying CSS, and CSS in general, you might also want to refer to:

- Books
  - Clayberg and Rubel, *eclipse: Building Commercial Quality Plug-ins* and the 2008 update *eclipse Plug-ins* were very good, but details are now out of date.
  - McAffer, Lemieux and Aniszczyk, *Eclipse Rich Client Platform* is a 2010 update.
  - McAffer, VanderLei and Archer, *OSGi and Equinox* concentrates on the plugin architecture at the basis of Eclipse.
- The Eclipse IDE online help section “Platform Plug-in Developer Guide”
- The CSS wiki on http://sourceforge.net/apps/trac/cs-studio/wiki.
- Finally, a Google search often gives good results because Eclipse/RCP is used all over the world by many developers.

Chapter 1. Introduction

There are fundamentally two ways to look at Control System Studio (CSS): As an end user, or as a developer and system administrator. This book is for the latter. It is organized in a way that it might help somebody who is about to install Control System Studio (CSS) at a site. It starts with a general overview of what CSS *is*, and then walks through the steps to compile pieces of CSS from source code, how to set up an archive engine, configure an alarm server and so on.

As an end user in the control room, you should be able to simply start CSS. In your office, you might be able to download CSS from some local web page for installation onto your office PC. In either case, the CSS product that you get has already been pre-configured for your site. As an end user, if you have questions about how to use CSS or some particular part of it, please refer to the online help that is accessible from within CSS via the menu `Help`, `Help Contents`. This book may still include some useful information about the details of a CSS installation, but the online help should be the primary source of information for end users.

Chapter 2. Control System Studio (CSS)

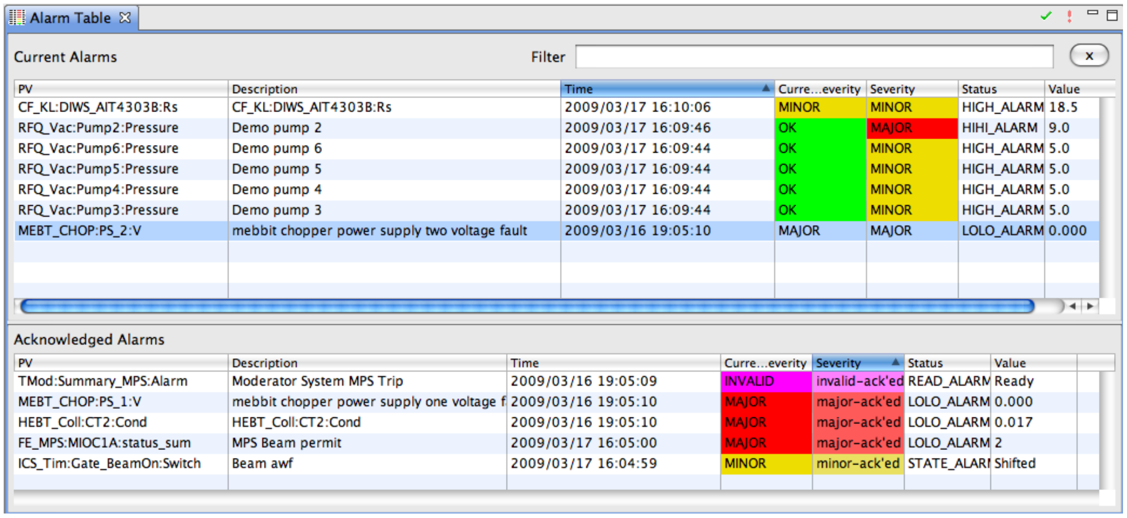
A first look at Control System Studio can be overwhelming. CSS is a collection of tools: Alarm handler, archive engine, as well as several operator interface and control system diagnostic tools. Most of them deal with Process Variables (PV), i.e. named control system data points that have a value, time stamp, alarm state, maybe units and display ranges, but they do this in different ways. One tool displays the value of a PV, one displays details of the PV configuration, while another concentrates on the alarm state of a PV. Each individual tool deserves some attention, and the Experimental Physics and Industrial Control System toolkit, EPICS, indeed offers each functionality as a separate tool. A key point of CSS is the integration of such functionalities.

To build a control system, one would typically select certain tools, configure them, deploy them in the control room, and then offer operators with some way of integrated access. For example, icons for the individual tools are placed on the desktop, or a Launcher application implemented in Python/TkInter is created to allow access to all control system tools from one top-level user interface. The same desktop computer used to access the control system might also run an EMail application, and most sites also have some type of Electronic Logbook, maybe with a web browser interface.

Integration of fundamentally separate tools via a Launcher still leaves you with stand-alone tools, running in parallel. CSS offers an integrated approach that might become more obvious in the following example scenario.

2.1. Example Use Case

Figure 2.1. CSS Alarm Table

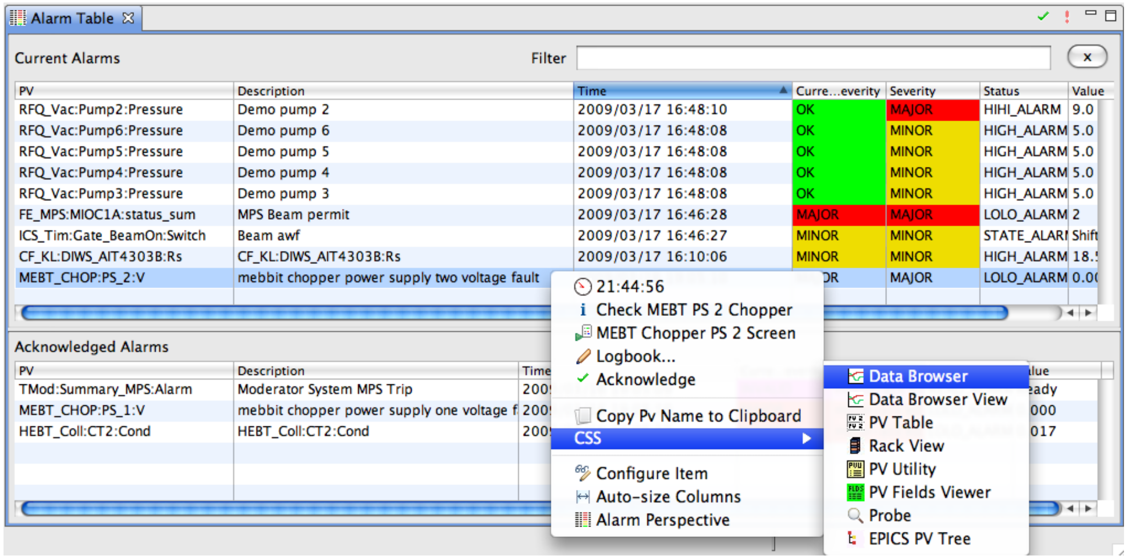


PV	Description	Time	Curre...everity	Severity	Status	Value
CF_KL:DIWS_AIT4303B:Rs	CF_KL:DIWS_AIT4303B:Rs	2009/03/17 16:10:06	MINOR	MINOR	HIGH_ALARM	18.5
RFQ_Vac:Pump2:Pressure	Demo pump 2	2009/03/17 16:09:46	OK	MAJOR	HIHI_ALARM	9.0
RFQ_Vac:Pump6:Pressure	Demo pump 6	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump5:Pressure	Demo pump 5	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump4:Pressure	Demo pump 4	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump3:Pressure	Demo pump 3	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0
MEBT_CHOP:PS_2:V	mebbit chopper power supply two voltage fault	2009/03/16 19:05:10	MAJOR	MAJOR	LOLO_ALARM	0.000

PV	Description	Time	Curre...everity	Severity	Status	Value
TMod:Summary_MPS:Alarm	Moderator System MPS Trip	2009/03/16 19:05:09	INVALID	invalid-ack'ed	READ_ALARM	Ready
MEBT_CHOP:PS_1:V	mebbit chopper power supply one voltage f	2009/03/16 19:05:10	MAJOR	major-ack'ed	LOLO_ALARM	0.000
HEBT_Coll:CT2:Cond	HEBT_Coll:CT2:Cond	2009/03/16 19:05:10	MAJOR	major-ack'ed	LOLO_ALARM	0.017
FE_MPS:MIOC1A:status_sum	MPS Beam permit	2009/03/17 16:05:00	MAJOR	major-ack'ed	LOLO_ALARM	2
ICS_Tim:Gate_BeamOn:Switch	Beam awf	2009/03/17 16:04:59	MINOR	minor-ack'ed	STATE_ALARI	Shifted

Let's assume an operator is running the `Alarm Table` shown in Figure 2.1, “CSS Alarm Table”. This application displays current alarms, and the operator notices a power supply voltage fault that needs further investigation. The alarm table indicates that this fault was reported by the Process Variable `MEBT_CHOP:PS_2:V`. Using separate tools, the operator could start a strip charting application, and enter the PV name into that other tool to review how the voltage changed over time. There may also be a way to copy and paste the PV name from the alarm table to the strip chart, but in any case the operator will first have to start the strip chart application.

Figure 2.2. Context Menu



Current Alarms

Filter

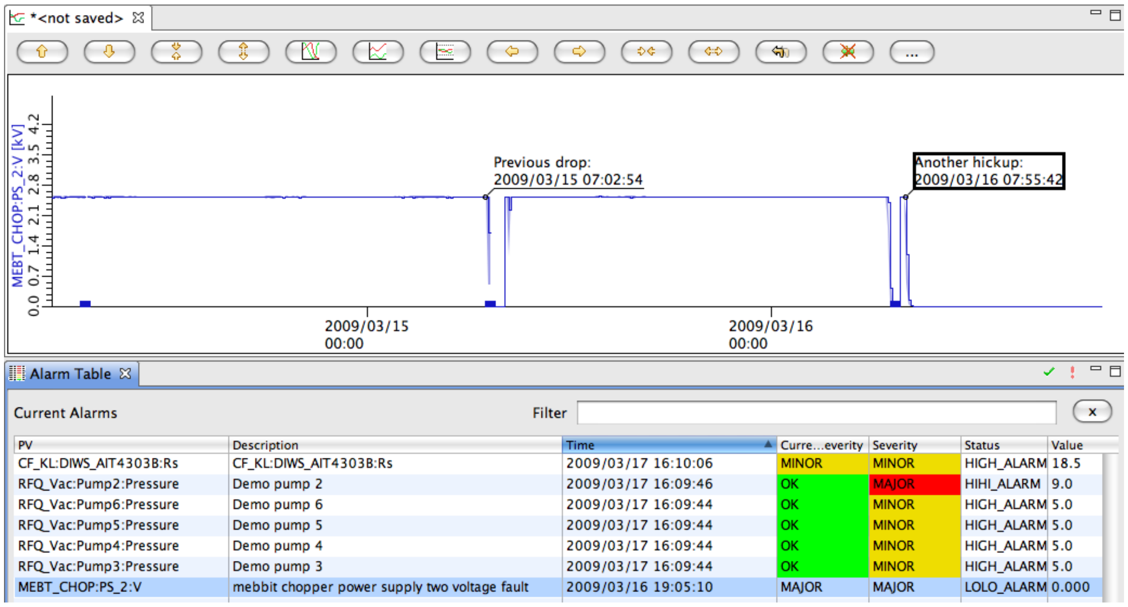
PV	Description	Time	Curre...everity	Severity	Status	Value
RFQ_Vac:Pump2:Pressure	Demo pump 2	2009/03/17 16:48:10	OK	MAJOR	HIHI_ALARM	9.0
RFQ_Vac:Pump6:Pressure	Demo pump 6	2009/03/17 16:48:08	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump5:Pressure	Demo pump 5	2009/03/17 16:48:08	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump4:Pressure	Demo pump 4	2009/03/17 16:48:08	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump3:Pressure	Demo pump 3	2009/03/17 16:48:08	OK	MINOR	HIGH_ALARM	5.0
FE_MPS:MIOC1A:status_sum	MPS Beam permit	2009/03/17 16:46:28	MAJOR	MAJOR	LOLO_ALARM	2
ICS_Tim:Gate_BeamOn:Switch	Beam awf	2009/03/17 16:46:27	MINOR	MINOR	STATE_ALARI	Shift
CF_KL:DIWS_AIT4303B:Rs	CF_KL:DIWS_AIT4303B:Rs	2009/03/17 16:10:06	MINOR	MINOR	HIGH_ALARM	18.!
MEBT_CHOP:PS_2:V	mebbit chopper power supply two voltage fault		R	MAJOR	LOLO_ALARM	0.00

Acknowledged Alarms

PV	Description	Time	lue
TMod:Summary_MPS:Alarm	Moderator System MPS Trip	200!	eady
MEBT_CHOP:PS_1:V	mebbit chopper power supply one voltage f	200!	000
HEBT_Coll:CT2:Cond	HEBT_Coll:CT2:Cond	200!	017

Enter CSS context menus: When right-clicking on the alarm under investigation in the alarm table, a context menu opens up, see Figure 2.2, “Context Menu”. It lists entries specific to this alarm, for example it allows the operator to acknowledge the alarm. In addition, there are links to other CSS tools, including the `Data Browser`. The Data Browser is a strip-charting tool for CSS. Selecting this Data Browser entry will open a new Data Browser plot and add the PV associated with the alarm to the Data Browser. With CSS, the task of opening another tool, then copying and pasting a PV name or even manually entering it has been reduced to a mouse click in the context menu!

Figure 2.3. Data Browser



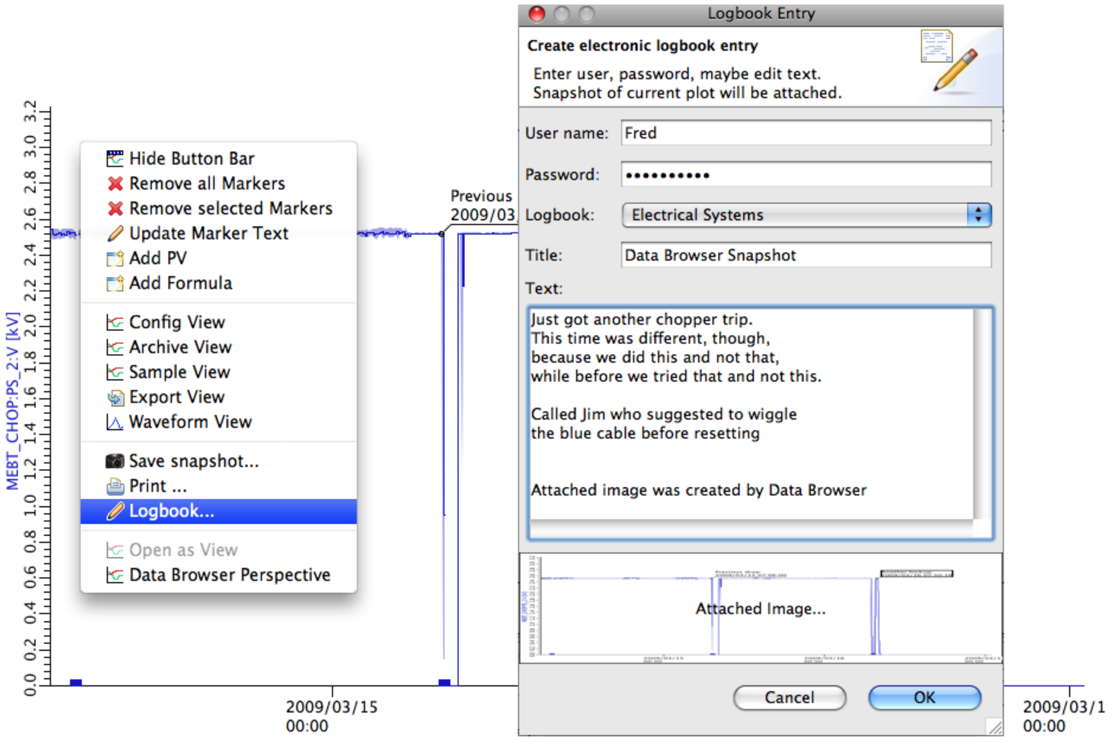
Current Alarms

Filter

PV	Description	Time	Curre...everity	Severity	Status	Value
CF_KL:DIWS_AIT4303B:Rs	CF_KL:DIWS_AIT4303B:Rs	2009/03/17 16:10:06	MINOR	MINOR	HIGH_ALARM	18.5
RFQ_Vac:Pump2:Pressure	Demo pump 2	2009/03/17 16:09:46	OK	MAJOR	HIHI_ALARM	9.0
RFQ_Vac:Pump6:Pressure	Demo pump 6	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump5:Pressure	Demo pump 5	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump4:Pressure	Demo pump 4	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump3:Pressure	Demo pump 3	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0
MEBT_CHOP:PS_2:V	mebbit chopper power supply two voltage fault	2009/03/16 19:05:10	MAJOR	MAJOR	LOLO_ALARM	0.000

In the Data Browser, the operator can now inspect the behavior of the PV over time. She might notice that the power supply voltage indeed dropped down, causing the alarm. In addition, she can see that the same voltage drop has occurred before, about one day ago. The operator can add annotations to the plot to indicate the time of the current as well as previous voltage drop.

Figure 2.4. Electronic Logbook Submission



User name: Fred

Password: ••••••••••

Logbook: Electrical Systems

Title: Data Browser Snapshot

Text:

Just got another chopper trip.
This time was different, though,
because we did this and not that,
while before we tried that and not this.

Called Jim who suggested to wiggle
the blue cable before resetting

Attached image was created by Data Browser

Eventually, the operator handles the alarm: Whatever caused the voltage to drop is understood and fixed. Especially when a similar event has happened before, as a day ago in this example, it makes sense to add a note to the electronic logbook, or to send an email to the engineer who maintains the power supply. If a site that uses CSS has an electronic logbook system (ELog), for example with a web interface, one could save a screen-shot of the Data Browser plot to a file, then go to a web browser, log into the ELog, describe the problem, attach the Data Browser screen-shot, submit the entry. With CSS, this can again be integrated: The context menu of the Data Browser plot directly offers a `Logbook...` link, see Figure 2.4, "Electronic Logbook Submission". When activated, a basic text entry with attached Data Browser screen-shot is prepared. The operator can enter the text, provide the user name and password that she uses in the ELog system, and submit the entry right from within CSS.

In summary, the whole work flow for handling an alarm that would otherwise require an operator to start separate stand-alone applications can be integrated within CSS:

1. View Alarms in the CSS Alarm Table.
2. Open Data Browser on an alarm from within the Alarm Table.
3. Send screen-shot of Data Browser plot, with explanation, to an electronic logbook.
4. Finally, acknowledge alarm in Alarm Table.

There is no need to copy/paste PV names, there is no need to save a screen-shot as an image file, remember the location of that file, then open another application to attach the file to an ELog entry.

2.2. Java, Eclipse, RCP

The functionality of Control System Studio (CSS) that we described in the previous example is implemented in Java, using the Eclipse software framework, specifically the Rich Client Platform (RCP).

The Java programming language and runtime environment allows the creation of software that can be used on several operating systems like Microsoft Windows, Linux and Apple Mac OS X. The same CSS application code can thus run in the control room as well as on office computers. While the Java runtime might in certain cases be slower than a program that was specifically created for a certain operating system in a language like C++, the speed is usually "good enough", and there are more advantages:

- Standard library covers basic data structures (lists, hash tables, ...) as well as network communication for all currently used protocols. No need to re-implement the wheel.
- Excellent software development tools: Debuggers, Profilers. Even without a debugger one can usually get a meaningful stack trace from a program that appears to be hung by sending the 'QUIT' signal to it (at least on Linux and OS X).

It is hard to imagine that a program suite as complex as CSS could have been implemented without Java.

As Java code becomes more complex, it can be split into several library files called Java Archives or 'JAR' files. The Java runtime can dynamically load and unload these libraries. Through introspection it is possible to locate features in such JAR files. For example, one can create a Java program that communicates with a control system, where the specific network protocol implementation is in a JAR file. While developing or later extending the software, a test JAR file is used to simulate a control system. For the operational setup, a site-specific JAR file connects the software to the actual control system. The same program can be used at different sites because the site-specific portion of the code is "plug-able".

While Java supports such dynamic binding, it does not enforce a standard way to do this. The Eclipse software framework provides three key elements:

1. `Plug-Ins`: A Plug-in is fundamentally a JAR file that the Eclipse runtime can dynamically load or also unload. Each Plug-in contains a `MANIFEST.MF` file that describes *dependencies*, i.e. which other Plug-ins are required to load a given Plug-in. Eclipse will automatically load these other Plug-ins as needed. The manifest file further defines which Plug-in content should be visible to other Plug-ins, and what is only accessible to code within the same Plug-in.
2. `Extension Points`: A Plug-in can define interfaces called extension points. An example would be an interface for getting data from a control system. Other Plug-ins can then implement them. The Eclipse *Registry* allows Plug-ins to locate available extension points.
3. `Rich Client Platform (RCP)`: Finally, Eclipse provides a complete *application framework* that is based on extension points. The entry point of the application itself, i.e. the "main" routine of the application, is an extension point. Items that are meant to appear in the menu bar are contributed by Plug-ins via extension points.

An Eclipse `Product` combines selected plugins with configurations files and a `Launcher`. Traditionally, Java products require a Unix shell script or a Windows batch file to set certain environment variables, configure the Java `CLASSPATH`, and finally invoke the Java runtime. Eclipse provides each product with a Launcher, which is an application native to the operating system. The Launcher can display a “Splash” screen, locate the Java Runtime, configure it, and finally start the product. To the end user, an Eclipse product thus looks just like any other application that is native to the operating system. CSS has icon and shows up as CSS in the task bar or process list, while traditional Java programs often appeared as shell scripts. While almost all the CSS and Eclipse Java code is fully portable across operating systems, this Launcher is specific to an operating system.

Another part that is operating system specific is the Eclipse Standard Window Toolkit (SWT). Java itself provides the Abstract Window Toolkit (AWT) to generate user interface code. At least in the past, AWT had an appearance that clearly differed from the native user interface of the operating system on which a Java application ran. The Eclipse community developed SWT, which always uses the native user interface elements of an operating system. For example, SWT on Mac OS uses `Cocoa` widgets, while SWT on Linux uses `GTK` widgets. The Eclipse and CSS code mostly uses SWT in a transparent way, but when building a product for a certain operating system, Eclipse includes the SWT plugins that are specific to that operating system.

2.3. RCP and Control System Studio

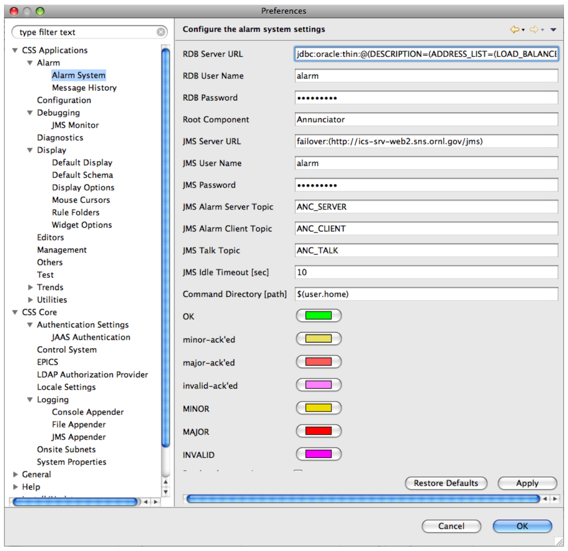
Control System Studio is an Eclipse RCP product, i.e. fundamentally just a collection of Eclipse Plug-ins. In CSS, what could otherwise be a stand-alone control system application turns into a Plug-in, for example:

- Probe Plug-in: Displays value of one Process Variable
- EPICS PV Tree Plug-in: Displays the input/output link hierarchy of EPICS records, for example a `calc` record's input links, which themselves might have input links and so on.
- Data Browser: Displays strip-chart type plots of Process Variable values over time

In principle, these could each be separate programs. Users would individually start and stop them as needed. In CSS, they become Plug-ins of the CSS product. Users run CSS, then start Probe or the Data Browser from within CSS. Just as a site integrator might choose which individual programs to install in the control room, she can add Plug-ins to CSS or remove them from CSS based on local requirements.

As a side note, there still *are* individual applications: The Alarm Server, the Archive Engine are examples for stand-alone RCP applications that use essential CSS library code, but they are nevertheless executed as individual application instances. In the following we are concentrating on the CSS product that end users see in the control room or on their office computers, where they typically run one instance of CSS and inside that includes Probe, EPICS PV Tree, Data Browser and other CSS tools.

Figure 2.5. CSS Preference Panel



Configure the alarm system settings

RDB Server URL	jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE
RDB User Name	alarm
RDB Password	••••••••••
Root Component	Annunciator
JMS Server URL	failover:(http://ics-srv-web2.sns.ornl.gov/jms)
JMS User Name	alarm
JMS Password	••••••••••
JMS Alarm Server Topic	ANC_SERVER
JMS Alarm Client Topic	ANC_CLIENT
JMS Talk Topic	ANC_TALK
JMS Idle Timeout [sec]	10
Command Directory [path]	$(user.home)
OK	🟩
minor-ack'ed	🟨
major-ack'ed	🟥
invalid-ack'ed	🟪
MINOR	🟨
MAJOR	🟥
INVALID	🟪

So what are the advantages of a combined Eclipse/CSS product over individual Java applications? For one, the online help and preference settings of Probe, EPICS PV Tree, Data Browser and other CSS tools are integrated. As shown in Figure 2.5, "CSS Preference Panel", one preference panel, opened from the menu bar under `Edit/Preferences`, provides access to all settings: Alarm system connectivity used by the Alarm Table, Data Browser settings can be found under the `Trends` section, and also generic control system settings like the `EPICS` connection parameters under `CSS Core`.

If the alarm plugins were removed from the CSS product shown in Figure 2.5, "CSS Preference Panel", the related settings would simply disappear from the preference panel. Likewise, when additional plugins are added to the CSS product, their preference settings will appear in the panel. With standalone, pre-

Eclipse control system applications, the preference settings were provided via environment variables or configuration files. Details tended to differ for each application. With CSS, the same Eclipse preference system applies to all application plugins. More in this in Chapter 6, *Hierarchical Preferences*.

CSS similarly benefits from the Eclipse/RCP online help system. Whenever CSS plugins include online help pages, these pages become part of the online help system, with global seach support.

What distinguishes CSS plugins from generic RCP plugins is the use of control system specific extension points. CSS defines extension points for providing live or archived data, to submit text and images to an electronic logbook. CSS application plugins can be implemented as users of these extension points. Combined with site-specific implementations of these extension points, a product is created that will for example get live data from EPICS Channel Access, archived data from a MySQL database, and send submissions to an electronic logbook that keeps its data in an Oracle database.

CSS furthermore defines control system specific classes like a Process Variable. Combined with an RCP mechanism called `Object Contributions`, this results in the seamless workflow between applications described in Section 2.1, “Example Use Case”: In the Alarm Table, each line that displays alarm information also “is” a process variable. The Data Browser is a tool that understands process variables. It registers with Eclipse as an Object Contribution for context menues whenever a process variable is selected. The Alarm Table code is not at all aware of the Data Browser, it simply displays a context menu with alarm related entries like “acknowledge”. Eclipse detects that the currently selected alarm table row also represents a process variable. Eclipse adds the Data Browser entry to the context menu, and when the user invokes that entry, Eclipse will start the Data Browser with the process variable name.

Based on the Object Contribution mechanism, applications that deal with PV names can exchange them without having any knowledge of each other; they remain seperate plugins, possibly implemented by different people. When a new PV-aware plugin is added to a CSS product, it will automatically appear in context menues of all other plugins that provide PVs.

Another feature that Eclipse/RCP offers are online updates: The CSS product can be build such that users download it onto their office computer. When the local CSS administrator publishes software updates to a site-specific update site, CSS will detect this on the next startup and prompt the user to update to the latest version. Similarly, optional CSS components that only some user need can be placed on a site-specific update site such that these users add them to their CSS product as needed.

Chapter 3. Control System

CSS interfaces to a control system. It is predominantly a client to the control system, reading data from control system Process Variables. While CSS includes server tools like the Alarm Server, that in turn is again a client to the control system.

When installing or learning to use CSS, a certain familiarity with the control system is assumed. For example, you will need to know the names of process variables that CSS can read or write. You might want to create new process variables that can serve as alarm triggers.

Like other control system client tools, CSS can use the meta data that comes with PVs. A PV value usually includes not only the basic value, for example a number like 3.13, but also a time stamp, a status/severity (OK, alarm, error), and information that display tools can use (value range, alarm limits, units). You need to understand what meta data your control system provides, and how to configure it. For EPICS, this means you should be able to create simple EPICS databases and execute them with the `softIoc` command. In the absence of a real control system, a few initial steps with CSS will be possible by using simulated PVs like `sim://sine`. See Chapter 25, *PV Access - org.csstudio.utility.pv* for more on PVs.

Chapter 4. Compiling, Running, Debugging CSS

End users of CSS, i.e. people who run CSS on their office computer, should not have to compile CSS. They can down-load a version of CSS that is already configured for use in offices at their site from a local web site, and updates to CSS will automatically be down-loaded from that same web site.

To reach that stage, somebody at each institute that uses CSS obviously needs to prepare such a local down-load site, compile CSS with suitable settings and place the binaries on the web site. Compilation from sources is also required for those parts of CSS that the end-user does not see: Archive Engine, Alarm Server and tools to configure the archive and alarm system.

Fundamentally, the compilation of sources into stand-alone RCP products that you can then deploy onto other computers is easy:

1. Get Java and Eclipse
2. Get the CSS source code
3. Start Eclipse, import the sources
4. Open the `*.product` file for CSS, ArchiveEngine, ... and “Export” as a stand-alone binary.

Instead of going to the last step, i.e. exporting a product from within the Eclipse Integrated Development Environment (IDE) into a deployable product, you have more options.

While you are still developing and testing the version of CSS that you assemble for use at your site, you can run the product within the IDE. You can execute it without first exporting it into a stand-alone binary. Running a product within the IDE is the preferred method during development because it allows full source code debugging and minimizes the turn-around time from editing the code to executing the product. This is also what you should do when first trying to run CSS from source code.

Once the product is acceptable for use at your site, you export it from the IDE and install the result on the desired computers. This is what you typically do for tools like the Archive Engine, Alarm Handler or other non-end-user tools: Export and install.

While the product export from the IDE is initially very convenient, it remains a manual process. The “Headless Build” of a CSS product is a script-driven process that can automate the product compilation. It can also be used for automated integration builds. As mentioned in Section 2.2, “Java, Eclipse, RCP”, Eclipse products are operating system specific. Eventually, most sites will implement a headless built for their primary end-user products, because automated product generation is more repeatable, especially when creating several products for various operating systems. As a result of a headless build you typically get not only the binary products, like CSS for your site for Windows and Linux in 32 and 64 bit versions. In addition, you get a repository from which existing CSS products that are already installed at your site can down-load updates.

4.1. Prerequisites

You will need

- Java Development Kit (JDK). It should be the JDK, not just a java Runtime Environment (JRE). It should be the Sun/Oracle or Apple JDK. OpenJDK, the GNU Compiler for the Java (GCJ) and other environments are currently not fully compatible with the Sun/Oracle JDK.

- Eclipse IDE for RCP/Plugin Developers. This has recently been called “Eclipse for RCP and RAP Developers”. Other versions like “Eclipse IDE for Java EE Developers” can be very useful if you also plan to do Web/Tomcat type of work, but the RCP version of the Eclipse IDE is the one that includes the full RCP source code and online help.

Both the JDK and the RCP IDE might have to be of a specific version, ask other CSS developers for the currently supported versions. At the time of this writing it has to be Java Version 1.6 also known as Java 6, and Eclipse 3.6 also known as Helios.

4.2. Obtaining the Source Code

For your first steps in CSS, it might be best to obtain a *snapshot of the sources* from one of the CSS sites. Such a snapshot will include what you need to build the products used at that site. When you access the CSS source repository, you can see the latest version of all sources from all CSS sites, which can be overwhelming.

To obtain a source snapshot, follow the links to collaborator sites on http://sourceforge.net/apps/trac/cs-studio/wiki and look for their source down-load.

The complete CSS sources are in a shared Source Forge repository using the Mercurial version control system. The project web site is http://cs-studio.sourceforge.net/ You can view the sources here: http://cs-studio.hg.sourceforge.net/hgweb/cs-studio/

To work on the CSS source code, i.e. to be able to submit changes, several steps are necessary:

1. Obtain Source Forge account.
2. Contact one of the CSS developers to gain write access to the repository.
3. Become familiar with the Mercurial version control system. You can download it from http://mercurial.selenic.com/, which also offers nice introductions.
4. Install the HG Eclipse plugin which provides Eclipse “Team” support for Mercurial, see http://www.javaforge.com/project/HGE.

For details refer to the CSS wiki at http://sourceforge.net/apps/trac/cs-studio/wiki.

In the following, we assume you are either already familiar with working on a Source Forge-hosted project, or you obtained a snapshot of the source code by other means, for example via down-load from a site that uses CSS. If you down-load the complete source code from the Source Forge repository, you will receive a directory tree similar to this:

```
core/features/org.csstudio.core.feature
core/features/org.csstudio. ... .feature
core/plugins/org.csstudio.data
core/plugins/org.csstudio.logging
core/plugins/org.csstudio...
applications/features/org.csstudio.opibuilder.feature
applications/features/org.csstudio. ... .feature
applications/plugins/org.csstudio.opibuilder
applications/plugins/org.csstudio.sds
applications/plugins/org.csstudio.trends.databrowser2
products/NSLS2/plugins/...
products/NSLS2/...
products/SNS/plugins/...
products/SNS/product/org.csstudio.basic.epics.product
```

```
built/...
```

The source code is roughly organized into

- `core` plugins and features that are shared by many if not all sites that use CSS
- `applications` plugins and features that are several sites use
- `product` code that is specific to a site like NSLS2 or SNS

If on the other hand you unpack a source code snapshot from a specific site, the result might be a flat directory including only those sources used to build that site's products:

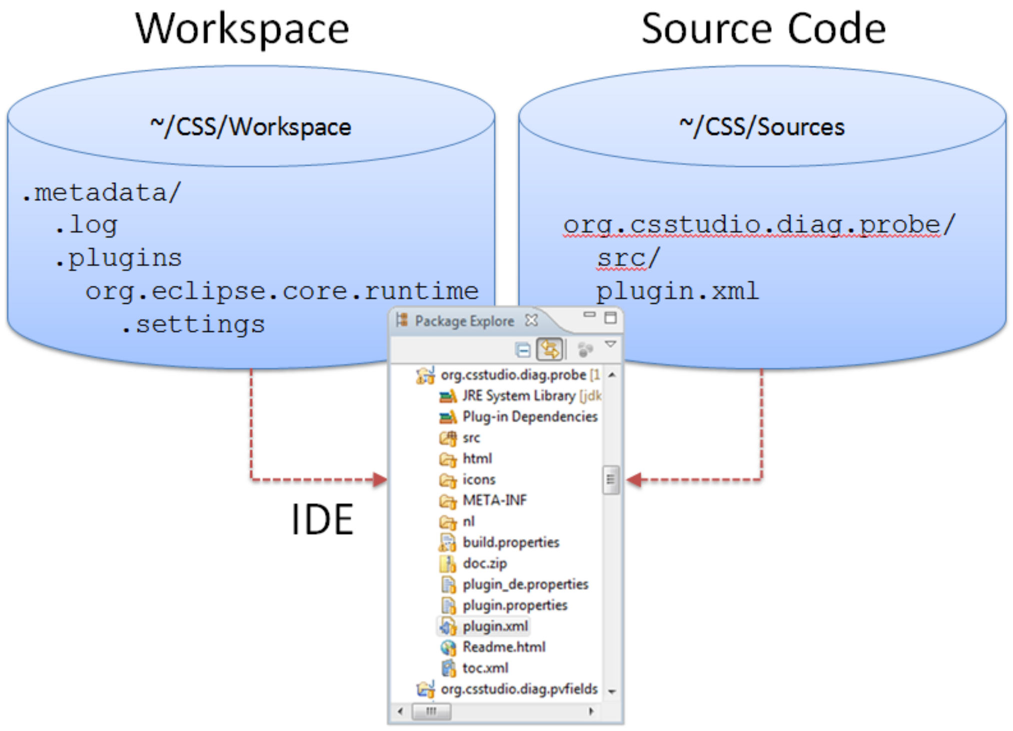
```
org.csstudio.data
org.csstudio.logging
org.csstudio...
org.csstudio.opibuilder.feature
org.csstudio.opibuilder
org.csstudio.trends.databrowser2
org.csstudio.basic.epics.product
```

4.3. IDE Example

Running any Eclipse RCP product from the Eclipse IDE is fundamentally simple: Open the product definition file, click “Launch ...”, done. The most difficult part is the initial Workspace setup.

When you start the Eclipse IDE for the first time, you will be prompted for a `Workspace`. This is the directory where Eclipse keeps configuration parameters, for example your compiler settings, in a `.metadata` subdirectory. The Workspace is *not necessarily* the same directory that holds the source files. Eclipse will in fact not recognize source files that are simply placed in the Workspace directory on your disk. Your source files need to be “imported” into the workspace to be recognized by Eclipse.

Figure 4.1. Workspace versus Source Code



In the following, we assume that you have the sources in a directory like `/usr/fred/CSS/Sources`, while your workspace is a separate directory `/usr/fred/CSS/Workspace`. To import the sources into the workspace:

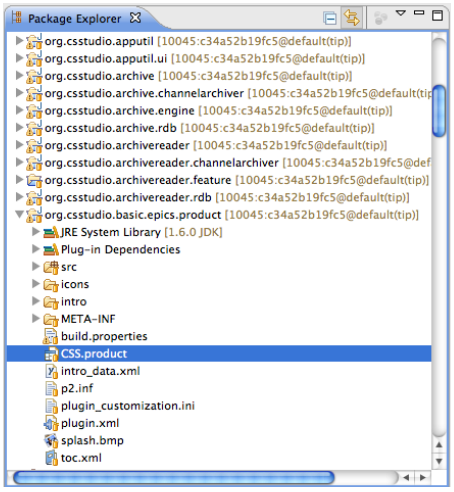
- In the IDE, select the menu `File/Import...`, `General`, `Existing Projects into Workspace`.
- As a root directory, select `/usr/fred/CSS/Sources`.
- *Do not check the option* "Copy projects into workspace"!
- By default all the plugins on the disk will be selected. You can leave them selected, or select only specific plugins as described below.
- Press `Finish`.

If you import from a source snapshot used by a CSS site, that snapshot might be limited to the plugins that the site-specific products use, and you can simply import all plugins. On the other hand, if you import from a full Source Forge repository checkout, you might want to un-select plugins that you know you will not need because otherwise you will end up with a workspace that is confusingly huge, even though you at best need only half of the plugins in there.

The problem might be that you have initially no idea which plugins you actually need. You can leave them all selected, then later remove those from the workspace that you find not to be needed for your products. When deleting plugin projects from the workspace, do *not* check the option to "Delete project contents on disk" which would delete the actual files. Instead, only delete the project from the workspace, but leave it on the disk and thus also in the software repository.

If you start with an initial selection of plugins that you assume you might need, look for errors about missing plugin dependencies, then import the missing plugins. Though this will initially require some trial and error, it might in the end be the most convenient way.

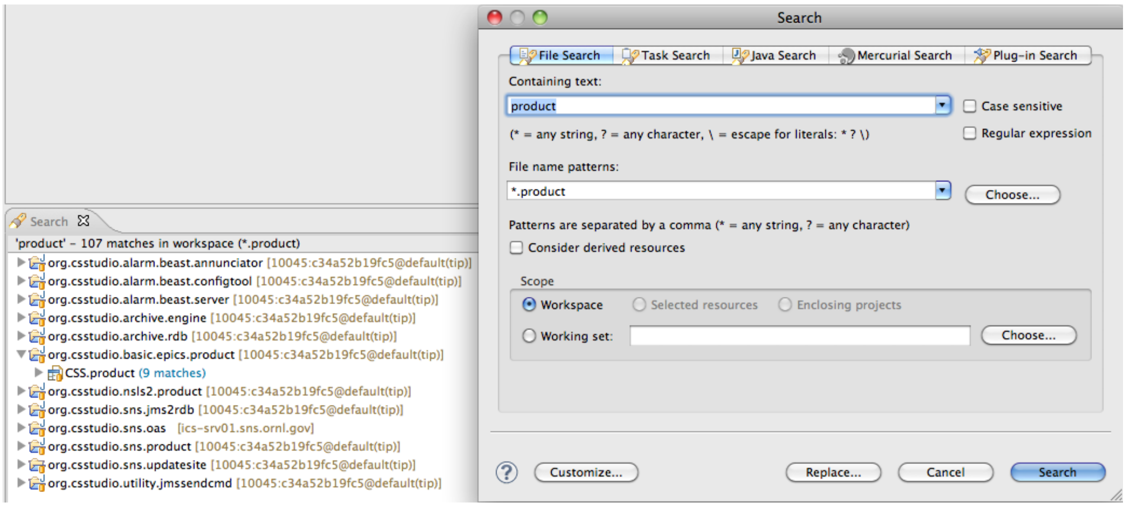
Figure 4.2. Eclipse Project Explorer



```
▶ org.csstudio.apputil [10045:c34a52b19fc5@default(tip)]
▶ org.csstudio.apputil.ui [10045:c34a52b19fc5@default(tip)]
▶ org.csstudio.archive [10045:c34a52b19fc5@default(tip)]
▶ org.csstudio.archive.channelarchiver [10045:c34a52b19fc5@default(tip
▶ org.csstudio.archive.engine [10045:c34a52b19fc5@default(tip)]
▶ org.csstudio.archive.rdb [10045:c34a52b19fc5@default(tip)]
▶ org.csstudio.archivereader [10045:c34a52b19fc5@default(tip)]
▶ org.csstudio.archivereader.channelarchiver [10045:c34a52b19fc5@def
▶ org.csstudio.archivereader.feature [10045:c34a52b19fc5@default(tip)]
▶ org.csstudio.archivereader.rdb [10045:c34a52b19fc5@default(tip)]
▼ org.csstudio.basic.epics.product [10045:c34a52b19fc5@default(tip)]
    ▶ JRE System Library [1.6.0 JDK]
    ▶ Plug-in Dependencies
    ▶ src
    ▶ icons
    ▶ intro
    ▶ META-INF
      build.properties
      CSS.product
      intro_data.xml
      p2.inf
      plugin_customization.ini
      plugin.xml
      splash.bmp
      toc.xml
```

The Eclipse `Project Explorer` should look similar to Figure 4.2, "Eclipse Project Explorer". It shows a flat list of plugins, even if the source directory tree has the plugins arranged into sub-directories like `core` or `applications`. It may take some time for Eclipse to compile all the sources. There should be no errors, i.e. in the package explorer there should be no red marks on any plugin in the Project Explorer.

Figure 4.3. Locating all Product Files


Search ☒
'product' - 107 matches in workspace (*.product)
- org.csstudio.alarm.beast.annunciator [10045:c34a52b19fc5@default(tip)]
- org.csstudio.alarm.beast.configtool [10045:c34a52b19fc5@default(tip)]
- org.csstudio.alarm.beast.server [10045:c34a52b19fc5@default(tip)]
- org.csstudio.archive.engine [10045:c34a52b19fc5@default(tip)]
- org.csstudio.archive.rdb [10045:c34a52b19fc5@default(tip)]
- org.csstudio.basic.epics.product [10045:c34a52b19fc5@default(tip)]
  - CSS.product (9 matches)
- org.csstudio.nsls2.product [10045:c34a52b19fc5@default(tip)]
- org.csstudio.sns.jms2rdb [10045:c34a52b19fc5@default(tip)]
- org.csstudio.sns.oas [ics-srv01.sns.ornl.gov]
- org.csstudio.sns.product [10045:c34a52b19fc5@default(tip)]
- org.csstudio.sns.updatesite [10045:c34a52b19fc5@default(tip)]
- org.csstudio.utility.jmssendcmd [10045:c34a52b19fc5@default(tip)]

Search

File Search | Task Search | Java Search | Mercurial Search | Plug-in Search

Containing text: product

☐ Case sensitive

(* = any string, ? = any character, \ = escape for literals: * ? \)

☐ Regular expression

File name patterns: *.product

Choose...

Patterns are separated by a comma (* = any string, ? = any character)

☐ Consider derived resources

Scope

● Workspace ○ Selected resources ○ Enclosing projects

○ Working set: Choose...

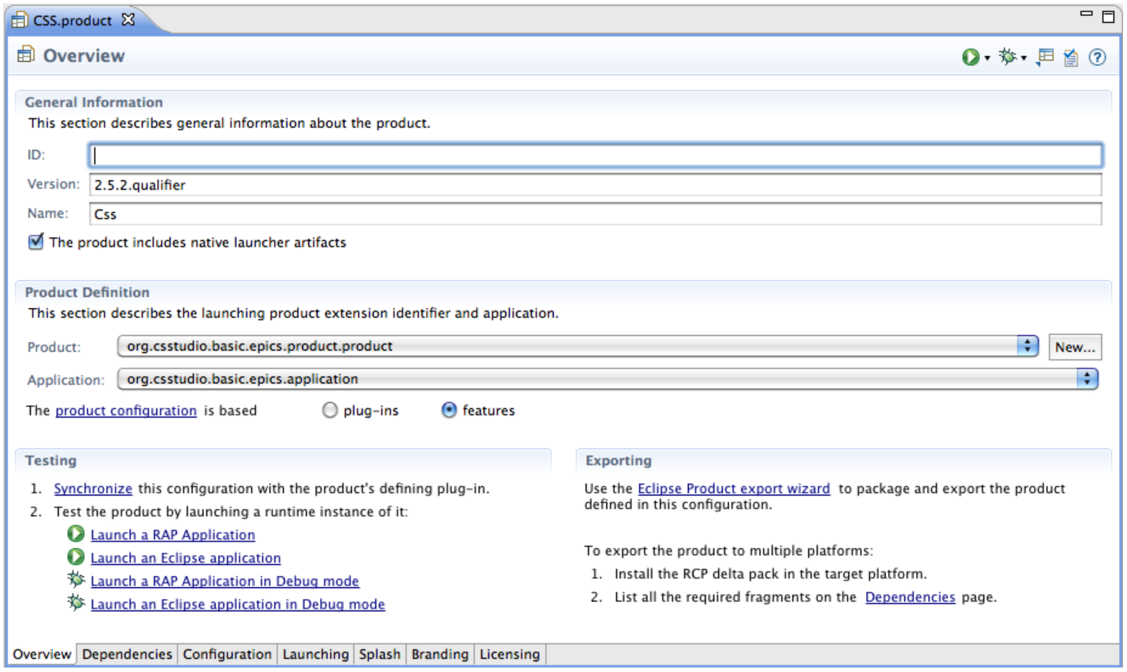
Customize... Replace... Cancel Search

Next you open one of the `*.product` files, for example

```
org.cs.studio.basic.epics.product/CSS.product
```

You can use the Eclipse `Search`, `File` menu to locate all available product files as shown in Figure 4.3, “Locating all Product Files”

Figure 4.4. Eclipse Product Editor


CSS.product ☒

Overview

General Information

This section describes general information about the product.

ID:

Version: 2.5.2.qualifier

Name: Css

☑ The product includes native launcher artifacts

Product Definition

This section describes the launching product extension identifier and application.

Product: org.csstudio.basic.epics.product.product New...

Application: org.csstudio.basic.epics.application

The product configuration is based ○ plug-ins ● features

Testing

1. Synchronize this configuration with the product's defining plug-in.
2. Test the product by launching a runtime instance of it:
   - Launch a RAP Application
   - Launch an Eclipse application
   - Launch a RAP Application in Debug mode
   - Launch an Eclipse application in Debug mode

Exporting

Use the Eclipse Product export wizard to package and export the product defined in this configuration.

To export the product to multiple platforms:

1. Install the RCP delta pack in the target platform.
2. List all the required fragments on the Dependencies page.

Overview | Dependencies | Configuration | Launching | Splash | Branding | Licensing

When you open an Eclipse product file, it will be displayed as per Figure 4.4, “Eclipse Product Editor”. In the `Overview` tab, first press `Synchronize`, then press `Launch an Eclipse Application` to run the product from within the IDE. The product should start up.

In the case of the Archive Engine it might soon exit with an error message indicating that it requires more command-line arguments. We will later describe how to “export” products from the IDE into standalone executables, and with such an exported command-line tool it will of course be trivial to provide command-line arguments when invoking the tool from the shell.

During development and initial testing, though, it can be more convenient to execute CSS products from within the IDE. To add command-line arguments to products executed in the IDE, invoke the menu `Run`, `Run Configurations...`. Locate the configuration for the product, open its “Arguments” tab. The “Program Arguments” section may already contain entries like `-os ${target.os}`. Add your desired command-line arguments to the end of the program arguments.

When running a product from within the IDE for the first time, the main goal is that the product should *start*. It may then stop because of missing command line arguments, but there should be no errors regarding missing plugins or compilation problems.

If the product does not start up, and instead you get an error like “Product ....could not be found” or “Missing required bundle ...”, it can be helpful to open the menu `Run`, `Run Configurations...` Locate the configuration that was automatically created when you tried to launch the product. On the `Plug-ins` tab, press `Validate Plug-Ins`. It should give a list of unresolved plugins, i.e. missing dependencies.

If you downloaded a consistent set of sources, you should have the missing plugins, but maybe you did not import them into the workspace. Import them, them add them to the run configuration by either manually selecting them or via the button `Add Required Plug-Ins`.

If you cannot find the missing plugins, there are usually two scenarios: The missing plugins are part of Eclipse, and you are using a version of Eclipse that is not compatible with the sources. For example, some plugin code may depend on `org.eclipse.core.runtime` version 3.6, but you are using Eclipse 3.5 which is too old. The other scenario would be that you obtained an inconsistent snapshot of the sources where some part is simply missing. In that case you probably need to contact the maintainer of the CSS product to determine why the product configuration has an error.

If you get the product to run: Congratulations! Almost as easy as running the product is debugging it: Instead of running the product again from the menu `Run`, `Run History`, you select `Run`, `Debug History` to start the product in debug mode. You can browse the source code, set breakpoints by double-clicking at the start of a source line, then step through the code from the breakpoint on.

4.4. JUnit Tests, Headless JUnit Tests

The Eclipse IDE has good support for executing JUnit tests and test-driven development in general. You can start many programming tasks by *first* implementing the JUnit test, using the IDE “Quick Fix” feature to create skeletons for the required classes and interfaces, which you then fill with the actual code until the test passes.

You will find several JUnit tests in the CSS plugin code because a lot of it was implemented in a test-driven fashion. These classes are often found in a `test/` source folder, and the source files will be named `*Test.java` or `*Demo.java`.

To execute a JUnit test, you simply right-click on the class file in the Project Navigator and select `Run As`, `JUnit Test`. Some tests will need configuration files to specify database URLs or other site-specific parameters. Refer to comments in the source code for details.

Eclipse supports a special type of JUnit test to allow testing within the Eclipse runtime environment. This is necessary for tests that depend on the Eclipse plugin registry and preference system. Tests that require the plug-in runtime should be in source files named `*PluginTest.java` or `*HeadlessTest.java`. To execute them, select `Run As`, `JUnit Plug-in Test` from the file context menu.

When you invoke such a JUnit Plug-in Test for the first time, the IDE will create a run configuration that loads all the plugins found in your workspace. If the plugin containing your test refers to Eclipse user interface plugins, the plugin test configuration will in addition start an instance of the complete Eclipse IDE with all your workspace plugins, then execute the test. This can take a long time and be overkill for what you want to test.

A “Headless” plugin test is a test that requires the Eclipse plugin runtime environment but *not* the complete IDE user interface. Such test source files are typically named `*HeadlessTest.java`. To invoke such a test, you can once execute it via `Run As`, `JUnit Plug-in Test` to create the basic run configuration, but then you should edit it as follows:

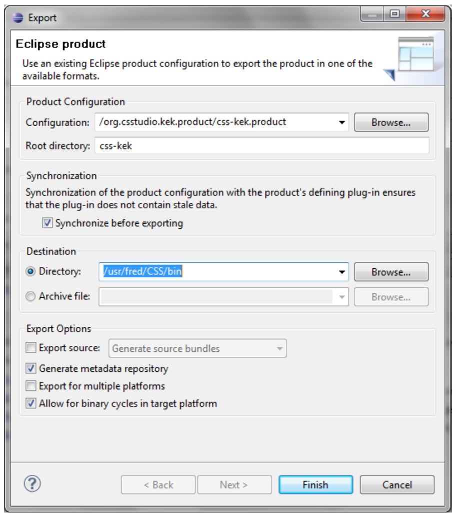
- Menu `Run`, `Run Configurations...`
- Locate the `JUnit Plug-in Test` that was created for your headless test.
- On the `Main` tab, select `Run an application` with the option `[No Application] - Headless Mode`.

  This will significantly reduce the startup time of your test because you avoid a copy of the Eclipse IDE.
- In the `Arguments` tab, you might need to add a program argument `-pluginCustomization /path/to/your/settings.ini` if your test needs certain preference settings.
- In the `Plug-ins` tab, by default all plugins in your workspace will be included. You can change that to only include selected plug-ins, which can reduce the startup time and might also be necessary to test the behavior of your test in case it depends on certain other plugins being available or not.

4.5. Product Export from IDE

End users of CSS cannot be expected to open the IDE, install the source code, and then run CSS from within the IDE. They need a product that executes from their “start” menu or a desktop link. One way to create such a standalone product is to “export” the product from the IDE.

Figure 4.5. Product Export Dialog



Export

Eclipse product

Use an existing Eclipse product configuration to export the product in one of the available formats.

Product Configuration

Configuration: /org.csstudio.kek.product/css-kek.product

Root directory: css-kek

Synchronization

Synchronization of the product configuration with the product's defining plug-in ensures that the plug-in does not contain stale data.

☑ Synchronize before exporting

Destination

● Directory: /usr/fred/CSS/bin

○ Archive file:

Export Options

☐ Export source: Generate source bundles

☑ Generate metadata repository

☐ Export for multiple platforms

☑ Allow for binary cycles in target platform

To export a product, open the product file that was already shown in Figure 4.4, “Eclipse Product Editor” and press the `Eclipse Product export wizard` link in the Overview tab. This will open the Eclipse product export dialog, see Figure 4.5, “Product Export Dialog”, where you enter the following:

1. As a Root Directory, enter the name of the product, for example ArchiveEngine, CSS, ... You could include a version number, for example CSS-3.0.

Use only a directory *name*, no path! This is the name of the directory that will contain the generated executable and associated files.

2. Enter a Directory *path* where the exported product should be placed, for example `/usr/fred/CSS/bin`

   Enter a full path for this option! In the end, the name from the previous option will be appended to this directory path, so your product will end up in a directory similar to `/usr/fred/CSS/bin/CSS-3.0`.

   Also assert that the directory that you selected is **empty**! When you export a product into a directory that already contains a product, maybe an older version of the same product, Eclipse will attempt to add the new code to the existing content, similar to an online update of an existing product. That is probably not what you want, so export into a an empty directory.
3. You might want to un-check the option to “Generate metadata repository”
4. Finish

You should now have a directory like `/usr/fred/CSS/bin/CSS-3.0` that contains a `css` executable. You can copy that directory to other computers and run it there. One method of deployment would be to offer a ZIP file of the exported product on a web page.

If you left the option to “Generate metadata repository” checked, Eclipse will generate also an update repository, see Chapter 17, *Update Repository*.

Command-line Products for Windows

Command-line applications, in Eclipse called “headless” RCP applications, are invoked from a terminal window, i.e. the Linux shell, Mac OS X terminal, or Windows Command Prompt. They are configured via command-line arguments, and they print information to the terminal.

Note that there is currently a limitation for command-line products on Windows! Headless RCP applications work fine on Linux and Mac OS X, but in the Windows command line tool `cmd.exe` you will by default *not see any output from headless RCP applications*. The problem is related to the fact that Java for Windows includes both `javaw.exe` and `java.exe` because Windows distinguishes between GUI and console applications. `javaw -version` invoked in `cmd.exe` will not display any output, either.

Eclipse for Windows likewise includes launchers `eclipse.exe` and `eclipsec.exe`. Headless products like an `ArchiveEngine`, `AlarmServer` etc. should use `eclipsec.exe` as their launcher, but there is currently no way to specify this in the product configuration, see also the bug report on “Support for exporting eclipsec.exe in RCP apps”, https://bugs.eclipse.org/bugs/show_bug.cgi?id=185205.

For the time being, the only solution is to manually replace the generated launcher, for example `AlarmServer.exe`, with a copy of `eclipsec.exe`:

```
copy \path\to\eclipsec.exe AlarmServer.exe
```

After replacing the original executable with a copy of `eclipsec.exe`, you can invoke the `AlarmServer.exe` from within the Windows Command Prompt, and you will see its command line output inside the Command Prompt window. If you start the `AlarmServer.exe` via double-clicks from the Windows Explorer, i.e. not from within a Command Prompt, it will actually open a new Command Prompt in which it can then display command line output.

4.6. Delta Pack, Cross-Platform Export

The product that you export from the IDE is by default limited to the operating system on which it was exported because of the OS-dependent launcher and SWT libraries, see Section 2.2, “Java, Eclipse, RCP”.

To export code for different platforms, you need the Eclipse “Delta Pack”. With the Delta Pack installed, Eclipse on OS X can for example build products for Windows, Linux, and OS X. Same for Linux and Windows.

To obtain the delta pack:

1. Goto the Downloads section on http://www.eclipse.org/.
2. Select “Projects”, “Eclipse Project”.
3. Select the version of Eclipse that you are using, for example 3.6.2.
4. Locate the Delta Pack, a file like `eclipse-3.6.2-delta-pack.zip`.

To use the delta pack:

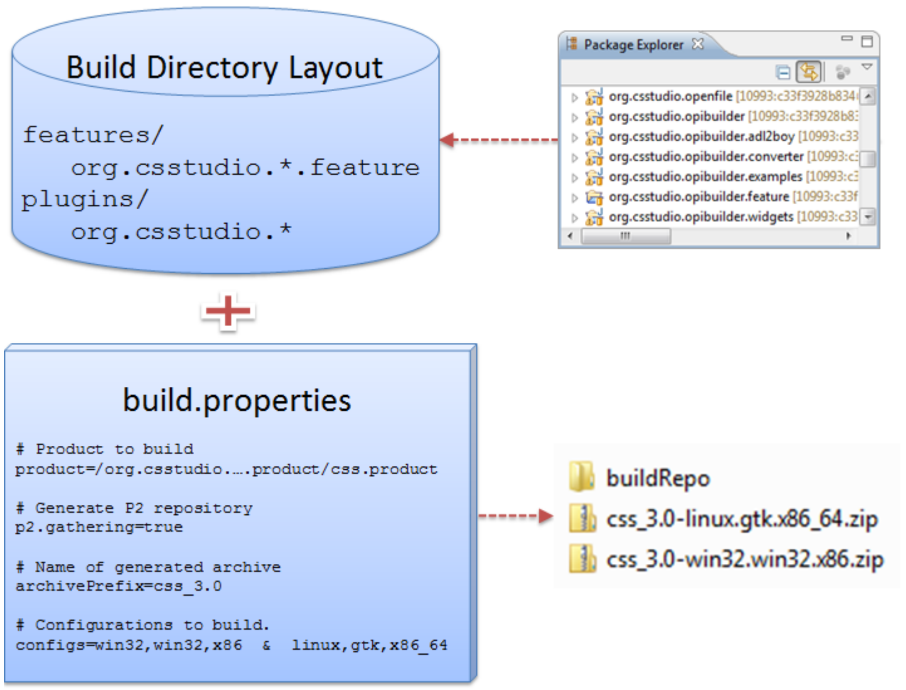
1. Extract the delta pack archive into its own directory on disk. The result should be an `eclipse` directory with sub-directories `plugins` and `features`
2. Open the Target Platform preferences: Menu `Preferences`, `Plug-in Development`, `Target Platform`. The Preferences item is usually in the `Windows` menu, but for Mac OS X it is in the `Eclipse` menu.
3. Edit the active target. Add an “Installation”, using the path to the delta pack's `eclipse` directory.

When you now export a product from the IDE, there will be a new option “Export to multiple platforms”.

4.7. Headless Build

The Headless Build is eventually necessary for each site that periodically publishes CSS product updates. It automates the build process, guaranteeing that the products are indeed exported for each supported platform. It can also be used as part of a nightly or integration build.

Figure 4.6. Headless Build


Build Directory Layout
features/
org.csstudio.*.feature
plugins/
org.csstudio.*
Package Explorer
org.csstudio.openfile
org.csstudio.opibuilder
org.csstudio.opibuilder.adl2boy
org.csstudio.opibuilder.converter
org.csstudio.opibuilder.examples
org.csstudio.opibuilder.feature
org.csstudio.opibuilder.widgets
build.properties
buildRepo
css_3.0-linux.gtk.x86_64.zip
css_3.0-win32.win32.x86.zip

```
# Product to build
product=/org.csstudio.….product/css.product

# Generate P2 repository
p2.gathering=true

# Name of generated archive
archivePrefix=css_3.0

# Configurations to build.
configs=win32,win32,x86  &  linux,gtk,x86_64
```

A headless build, however, is more complicated to set up than the product export from the IDE. The headless build does *not* use the workspace nor the graphical IDE. Instead, it is command-line driven and expects to find all feature and plugin projects in subdirectories of the build directory which have to be named `features` respectively `plugins`.

This usually requires a shell script or a build file for `ant` to copy all feature and plugin projects that are part of your product into such a build directory layout. In addition, you need a file `build.properties` to specify which product or feature to build, which target architectures to use, and how to name the generated archive files.

For details, check the Eclipse online help for Headless Build, or refer to the scripts that other CSS sites use to build their products, which can be found in the `build` and `products` subdirectories of the CSS sources.

4.8. Feature Patch

A side effect of the headless build and a P2-managed product is that it is no longer possible to simply replace plugins in a product with new versions. If you replace a plugin JAR file with a different version, even if the names exactly match, P2 will recognize the change because of checksums. It will refuse to load the modified plugin, because it was part of the original configuration. In the spirit of maintaining a well defined product with known content, this makes sense. Occasionally, however, it is a big nuisance: Fixes to small bugs are no longer possible via basic plugin updates. Instead, you have to create a new product by incrementing all version numbers, perform a complete headless build, then publish the new product and its repository on the update site.

A feature patch can be used to update only part of a product while maintaining full configuration control. Assume our current product contains a plugin `org.csstudio.trends.databrowser2` with version number 3.0.1 that we intend to replace with a newer version 3.0.2. We need to determine which feature provides that plugin in the product-to-update, and we need to determine its exact version, including a possible date/time qualifier. If you do not know which feature provided the original plugin, search the files in the `features` subdirectory of the installed product. Assume we find that the plugin was provided as part of the feature

```
org.csstudio.trends.databrowser2.feature_3.0.1.20110715
```

Create a new Feature Patch project in the Eclipse IDE:

- As a project name you can for example use “databrowser3.0.1.patch”.
- In the “Properties of feature being patched” section of the project wizard enter the “Feature ID” of the original feature, i.e. `org.csstudio.trends.databrowser2.feature`.
- Under “Feature Version” be sure to enter the correct version that you intend to update, i.e. `3.0.1.20110715`.
- Add the plugins that should be updated with the patch, i.e. `org.csstudio.trends.databrowser2`. The feature that we are patching might contain many more plugins, but in the patch we only include what we want to add or replace. Assert that the version numbers of all the plugins that you want to replace have been incremented from their installed version.

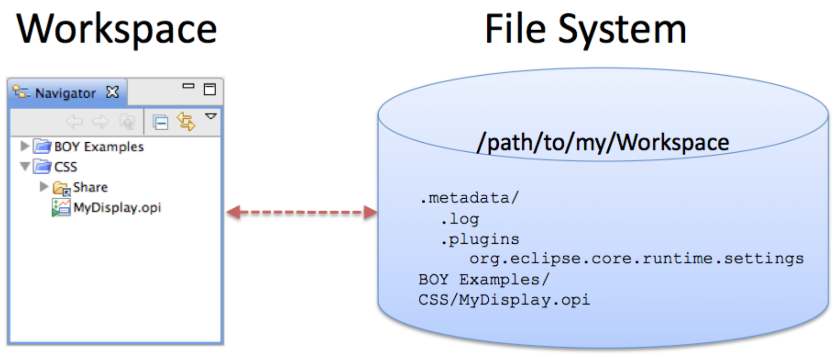
You can now export the feature patch, using an archive file like `databrowser3.0.1.patch.zip` as the target, and then use that ZIP file to install the patch into your product via the menu item `Help`, `Install New Software...` by adding the ZIP file to the available software sites.

As a result, the original `org.csstudio.trends.databrowser2` plugin is replaced with the new one, while P2 remains fully aware of what version of which plugin was installed from where, so it will for example allow you to un-install the patch, or later add additional feature patches.

Chapter 5. Workspace

Eclipse products use a “Workspace”. This is a directory that stores your configuration files and settings, allowing Eclipse to start up in the same state as you left it when it was running the last time. You can only run one instance of Eclipse for a selected workspace.

Figure 5.1. Workspace



In the previous chapter, we already used the workspace within the IDE. We described the relationship between the actual location of CSS plugin source code and the way these plugins appear in the IDE. Figure 5.1, “Workspace” shows the similar relationship between the workspace of a CSS end-user and the actual file system directory. Directories and files in the workspace directory are visible in the `Navigator` view, for example display files or Data Browser configuration files. Double-clicking on these files will open them. The actual workspace location and name of the workspace, i.e. the `/path/to/my/Workspace` in the example, is usually hidden from the end user. Likewise, the `.metadata` directory where Eclipse stores user preferences, the current location of windows etc. are hidden from the user so that she can concentrate on just the files of interest like display files.

5.1. Selecting a Workspace

Every Eclipse RCP application allows you to set the workspace when starting the product from the command-line via the `-data` option:

```
the_css_product -data /path/to/my/Workspace
```

Most products also offer a menu item `File`, `Switch Workspace...` that displays the current workspace location in the file system and allows you to select a different one.

Some products like the Eclipse IDE and the SNS version of CSS have a startup dialog that prompts users for the workspace.

5.2. Log File

One important file within the workspace is `.metadata/.log`. Note the leading dot on both the metadata directory and the log file which makes this a hidden file on Linux and Mac OS.

This file contains Eclipse log messages. In case of problems it is often useful to look for error indications in the log file. When you start the product with the folloing command-line option, you will also see log messages on the console:

```
the_css_product -consoleLog
```

5.3. Projects, Saving Files, Default Project

The top-level elements of a Workspace are called “Projects”. They can be created via the menu `File`, `New...`.

Each workspace must have at least one such project, because otherwise you cannot save any files. Many versions of CSS therefore create a default project called “CSS” in case it does not exist already when CSS is started up.

5.4. Linked Folders

Projects and sub-folders of a project in a user's Workspace are private to that user. If you want to share configuration files with other CSS users, for example use common operator interface panels or Data Browser configurations, you can link to shared folders in the file system.

To manually create a linked folder:

- Open the `Navigator` view.
- Right-click on an existing project or folder.
- Select `New...`, `Other...`, `General`, `Folder`, press `Next`.
- Enter the desired folder name, but do *not* press `Finish`, yet!
- Press the `Advanced...` button to select a “Link to alternate location (Linked Folder)”, browsing to the desired location outside of your workspace.
- Now press `Finish`.

Linked folders can also be generated via command-line options. This allows sites with a shared file system to automatically include suitable links to these shares in every CSS workspace by invoking CSS from a script with the following option:

```
-share_link /var/x/y=/CSS/Share,/var/x/z=/Project/Folder/Link
```

The `-share_link` option takes one or more comma-separated values. Each value is of the form `path=resource`. The “path” represents a path to a folder in the file system, and “resource” is the resource to create within the workspace. If they include spaces, the file system path and resource must each be enclosed in double-quotes.

Examples:

`-share_link /path/to/share`: If only the file system path is provided, the resource defaults to “/CSS/Share”, i.e. a linked folder “Share” within the project “CSS” will be created that links to `/path/to/share` in the file system.

`-share_link /path/to/share,"/path/to/an other"=/MyProject/Other/Share`: As before, and in addition a project “MyProject” will be created with folder “Other”, and finally a linked folder “Share” is created within that folder which points to `/path/to/an other` in the file system. The file system path must be enclosed in double-quotes because it contains a space.

Note that linked folders in the workspace behave similar to symbolic links in the Unix file system. If the file system path is not valid, the linked resource is still created but will appear empty. When opening its properties, the location will be marked as “Does not exist”.

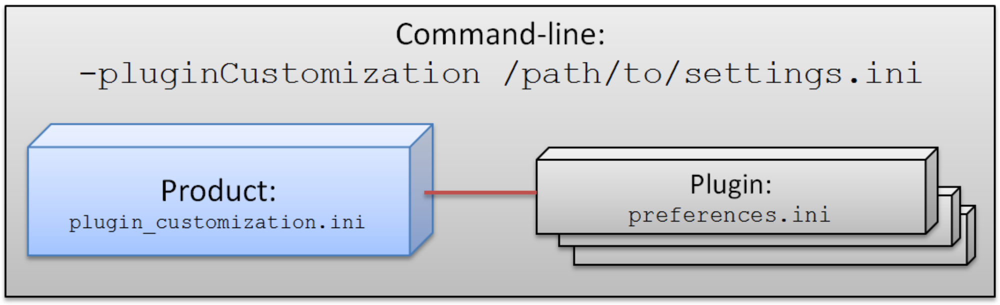
Chapter 6. Hierarchical Preferences

Once you are able to start a CSS product, you need to configure it. For example, to run the archive engine, you need to specify where it should write data. Likewise, the CSS Data Browser needs to be configured to read the archived data from that location. These settings are site-specific, they cannot be included in the source code. The local CSS administrator should be able to configure these settings so that local end users of CSS can simply open the Data Browser without having to adjust the archive data source settings. Other settings might have useful defaults, but end users may still prefer to adjust them to their liking.

Eclipse has a preference system that allows each plugin to provide default settings. The builder of a product can then add site-specific settings, and finally the users are able to adjust them individually for their instance of CSS.

Note that the following is a technical view of preference settings: How they are originally defined and then adjusted on several levels. This is the information that CSS developers need. As an end user, you might want to read this *from the back*, jumping right away to Section 6.4, “End User Settings: Preference Pages”. As somebody who downloaded some generic version of CSS and wants to provide all the settings for your site, you would follow Section 6.3, “Command-line Adjustments: -pluginCustomization”.

Figure 6.1. Hierarchical Preferences



6.1. Plugin Defaults: preferences.ini

Each plugin with configurable settings should have a file `preferences.ini` in its root directory. This file has several purposes:

- It defines those settings, i.e. their name.
- It documents their meaning, explaining supported values.
- It establishes the default value for each setting.

For example, such a file can be:

```
# Example preferences.ini for plugin org.csstudio.demo

# Enable the super feature.
# Set to 'on', 'off' or 'automatic' which enables it whenever possible
enable_super=automatic

# URL of server where the super feature connects
url=http://localhost/superdata
```

The Eclipse preference service for a plugin will automatically use such a file as long as it has the correct name and is located in the plugins root directory. So for a pluging called `org.csstudio.x.y`, that file must be called `org.csstudio.x.y/preferences.ini`.

Note that Eclipse offers more ways to establish defaults, but they are discouraged for CSS because of disadvantages:

- Defaults hard-coded in calls to the preference service: These are difficult to find in the code.
- Preference Initializer extension point defined in plugin.xml: This again hides the preference tags and their default values in the code.

Putting the defaults into `preferences.ini` seems the best way to define *and* at the same time document them.

6.2. Product (Site) Defaults: plugin_customization.ini

When bundling plugins into a site-specific product, a file `plugin_customization.ini` in the plugin that defines that product is automatically used by Eclipse to override settings from individual plugins.

```
# Example plugin_customization.ini

# Override defaults of plugin org.csstudio.demo:
# URL of our super feature server
org.csstudio.demo/url=http://my.site.org/mysuperdata
```

Note that in comparison to an individual plugin's preference file, all settings in this global preference file are prefixed with the name of the affected plugin. If you wonder about the names of preference settings supported by each plugin, consult the `preferences.ini` of the respective plugin.

The file has to be in the plugin that defines the product. For example, assume you have a product X that is defined in `org.csstudio.x.y/X.product`. Eclipse will look for a file `org.csstudio.x.y/plugin_customization.ini`, i.e. a customization file in the root directory of the plugin that defines the product, for settings to override values from individual plugins.

In the compiled version of the product, plugins are usually JAR files in a `plugins` sub-directory of the installed product. The customization file will then be found inside `plugins/org.csstudio.x.y_*.jar` where the `*` represents some version number and date. End users will not open and modify that JAR file and the `plugin_customization.ini` file inside the JAR. This way, your product has the settings for your site “built-in”.

In principle, you *could* actually un-zip the JAR file, edit the customization file, and re-zip it. As a CSS maintainer for your site, you may occasionally be tempted to do this. The downside is that such changes are easily forgotten, and the next time you build a product, you would have to re-do the un-zip, edit, re-zip hack. A better approach to applying changes to the build-in settings of a product is described in the next section.

6.3. Command-line Adjustments: -pluginCustomization

If a select installation at a site requires a few extra changes, for example a test network installation needs settings that differ from your main campus network, put those into a file with the same format as `plugin_customization.ini`, for example `testnet.ini`, and run the product with a command-line option

```
my_product -pluginCustomization /path/to/testnet.ini
```

While that customization file has the same format as `plugin_customization.ini`, its name and location are arbitrary. You can call it anything and place it anywhere, as long as you provide the full path to it via the -pluginCustomization command line option.

In fact you want to **always provide the full name** to the file, even if it is in what you consider the current directory, because the Java runtime that is started by the Eclipse RCP launcher could have a very different idea of its current directory.

If Eclipse does not find your -pluginCustomization file, it is simply ignored, there will be no error message! So if the settings that you think you put into a -pluginCustomization file are not having the desired effect, triple check that you indeed provided the full and correct path to the file.

6.4. End User Settings: Preference Pages

Finally, the GUI code can offer Preference Pages via which the end user can change the settings. The exact location of the menu entry for opening the Preference Pages can change. Eclipse usually has a Preference entry in the `Window` menu, except on Mac OS X where it is placed in the `Eclipse` menu. CSS provides a Preference entry in the `Edit` menu regardless of the operating system because that seems to be a common location for the preference settings in other programs. (Older versions of CSS actually used the `CSS` menu)

Typically, these end-user settings are saved in the user's workspace, i.e. they persist for the current user but will not affect the settings of other users or even the site-wide defaults. Details depend on the preference store that is selected by the code that implements the preference page.

6.5. Secure Storage of Passwords

The CSS `SecureStorage` and `PasswordFieldEditor` allow to keep preferences for passwords encrypted. It is for example used to store passwords for accessing the relational database of the SNS alarm and archive system. These encrypted preferences can be stored in either of two places:

- `INSTALL_LOCATION`:

  This is the directory where CSS is installed, for example `/usr/local/css/CSS3.0.0`. Using the install location has the advantage that all users of CSS on that computer will have the password available for use without actually knowing it.

  The typical scenario is this:
  - System administrator installs CSS.
  - System administrator starts CSS and enters the passwords once in the preference GUI.
  - Users can now run CSS. CSS plugins have access to the password, but users cannot see the password.

  The disadvantage is that if a user were to try to change any of the related preferences from the GUI, CSS will try to write the password and fail because ordinary users have no write permission in the install location.
- `CONFIGURATION_LOCATION`:

  The exact directory name of this location is determined automatically based on where the user can write. It can for example be in an `.eclipse` sub-directory of the user home directory. Using the configuration location has the advantage that every user of CSS can enter his or her own password. The password they entered usually applies to their instance of CSS, but may also affect instances started by other users.

  The typical scenario is this:
  - System administrator installs CSS, maybe with some passwords build into the product's `plugin_customization.ini`.

- User can run CSS. CSS plugins have access to the built-in password.
- User knows another password, enters that password in the preference GUI.
- For that user, CSS plugins now use the password that the user entered.

The disadvantage of this approach it its unpredictability. The configuration location can default to the CSS installation location *if that user has write permissions* to that area of the file system. If such a user enters a password, it will be used by all instances of CSS on that computer. If a user with lesser file access privileges enters a password, it will only be written to a configuration location in that users home directory and *not* apply to other users.

The location for secure storage can be configured via this setting:

```
# Can be either "INSTALL_LOCATION" or "CONFIGURATION_LOCATION".
# By default it is "CONFIGURATION_LOCATION".
org.csstudio.auth/secure_storage_location=CONFIGURATION_LOCATION
```

org.csstudio.sns.passwordprovider

The Eclipse `SecurePreferencesFactory` that is used for the secure storage of passwords as described above requires a password provider. This password is sued to encrypt and descrypt the preferences. Eclipse provides implementations for Windows or OS X that can interface with OS-specific key stores, including

- org.eclipse.equinox.security.macosx
- org.eclipse.equinox.security.win32.x86

On other operating systems, Eclipse will fall back to a dialog box that queries the user for a master password. The plugin `org.csstudio.sns.passwordprovider` provides a password without requiring user input, which can be very convenient. Since the password can be derived from reading the code, it is not 100% secure

Chapter 7. Console

All Eclipse RCP programs, and this includes the CSS end-user GUI as well as command-line tools like the archive engine, alarm server etc. include the OSGi console.

The console allows you to view which plugins have been loaded, to introspect extension points and more. In principle it can be used to load additional plugins or to replace existing plugins at runtime, but at this point no CSS tool is using that feature.

7.1. Enabling the Console

To start any RCP program with console access, add the command-line option

```
-console
```

You can allow network access to the console by adding a TCP port number, for example

```
-console 4884
```

You will then be able to access the console of the program remotely via

```
telnet host_where_program_is_running 4884
```

Multiple network connections to the same program are possible. Note that there is no security, i.e. anybody on the network could “telnet” to your application and stop it!

7.2. Console Commands

Useful console commands:

- `help` - List all available commands.
- `ss` - List all plugins and their “short” status.
- `ns` - List all plugins that define extension points.
- `ns name.of.some.plugin` - List all extension points of that plugin.
- `pt name.of.some.extension.point` - List all implementations of that extension point.
- `disconnect` - Disconnect from a telnet session to the console. The application will continue to run.
- `close` - Close, i.e. stop the program. Note that this does not just end the console session, it stops the application. The `IApplication.stop()` method will be invoked, which allows for a graceful shutdown of the application.

There are also commands to list all applications, stop an application, update or add plugins, then restart the application. Refer to the `help` command for more.

7.3. Adding Commands

Applications can add custom commands to the console. To accomplish this, you need to implement an

```
org.eclipse.osgi.framework.console.CommandProvider
```

and register it with the `CommandProvider` service. This can be done from the `start()` method of the plugin activator:

```

class MyActivator
...
  public void start(final BundleContext context)
```

```
      throws Exception
  {
     // Register console commands for this engine
     commands = new MyConsoleCommands();
     context.registerService(CommandProvider.class.getName(),
                             commands, null);
     ...
```

When implementing your `CommandProvider`, note that there is no interface in the usual Java sense to declare your console commands. Instead, all public methods starting with an underscore in their name and a `CommandInterpreter` parameter will be available as console commands:

```

public class MyConsoleCommands implements CommandProvider
{
    @Override
    public String getHelp()
    {
        final StringBuilder buf = new StringBuilder();
        buf.append("---My commands---\n");
        buf.append("\thello - Say hello\n");
        return buf.toString();
    }

    /** 'hello' command */
    public Object _hello(final CommandInterpreter intp)
    {
        intp.print("Hello");
        return null;
    }
}
```

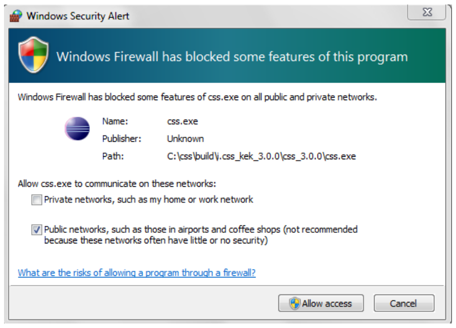
Chapter 8. Network Usage by CSS

As a control system tool, CSS naturally performs a certain amount of network communication. For the most part, the CSS application run by end users acts as a network *client*, while tools like the Alarm Server will obviously also need to *serve* data. CSS as a client tool can run without administrator privileges or firewall exceptions.

8.1. Windows Firewall Warning

Even when running CSS as a client tool, you might run into the Windows Firewall warning shown in Figure 8.1, “Windows Firewall Warning”.

Figure 8.1. Windows Firewall Warning



Windows Security Alert

Windows Firewall has blocked some features of this program

Windows Firewall has blocked some features of css.exe on all public and private networks.

Name:	css.exe
Publisher:	Unknown
Path:	C:\css\build\j.css_kek_3.0.0\css_3.0.0\css.exe

Allow css.exe to communicate on these networks:

☐ Private networks, such as my home or work network

☑ Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing a program through a firewall?

The message appears because even a client tool like CSS does sometimes *listen* on network ports, i.e. act as a server. This usually happens for the following reasons:

- You configured logging to JMS (see Chapter 23, *Logging - org.csstudio.logging* and Chapter 11, *Java Message Server*). While connecting to JMS, CSS will also listen.
- You opened the online help. Internally, this is implemented by CSS acting as a web server, and the help viewer is an ordinary web browser.

You can usually cancel that firewall warning, meaning Windows will **block** access from other computers on the network to your instance of CSS. Logging to JMS on another computer as well as local viewing of the online help will not be affected by the firewall blocking outside access to your CSS client.

8.2. Required Firewall Exceptions

You might have to open firewall exceptions for all the tools that *serve* data, including

- Your RDB server: Check ports used by your RDB.
- Your JMS server: Usually port 61616.
- Archive Engines: Allow access to the status web server that you configure for each archive engine instance.

For details on how to allow such accss, you will have to refer to the documentation of your firewall: Linux iptable, Windows firewall.

Chapter 9. Relational Database (RDB)

Several CSS tools interface to a relational database. The archive system described in Chapter 10, *Archive System* can store data in an RDB. The alarm system described in Chapter 13, *Alarm System* keeps its configuration and persistent state there.

To end users of CSS, this is mostly transparent, but if you install and administer CSS for your site, basic RDB administration skills will be required.

9.1. Supported Databases

CSS includes JDBC libraries for the following databases, and the alarm and archive system includes example database definition (DBD) files to create the required tables for these database dialects:

- MySQL: This database is often the easiest to set up for initial tests and to support smaller operational setups. Free, open-source.

  The Community Version 5 should be usable.
- Oracle: This database system might be the most powerful, with virtually unlimited table sizes. It is, however, not free.

  Oracle 10 and 11 should work.
- PostgreSQL: This database might be a good compromise. Bigger table sizes. Free, open-source.

  Postgres has to be at least version 8.4 to support sequences.

Your personal preference might of course differ, and it is also important to consider what database system is already supported at your site.

9.2. RDB User Accounts

You will need to configure at least three types of RDB users:

- Administrator account: A user that can create tables, update indices, reset sequences etc.
- Write-access users: The ArchiveEngine needs a user account that can write samples to the archive related tables. The AlarmServer needs a user account that can persist the alarm state. For each system you might want to create a user that can write to the necessary tables of that system, but only that system, and without general administrative rights.
- Read-only access: The CSS data browser needs a user that can read archived data. You might want to create various reports for the RDB data. All of these could use one shared read-only account, a name and password that you can freely distribute to end users who want to create their own reports on the data with MS Access, JSP, PHP, ...

9.3. Network Access

CSS or Java JDBC tools in general tend to connect to the database from the network. Even if CSS is running on the same computer that also hosts the database, it will still connect to `localhost` via network system calls.

You should therefore check that the database is network accessible. Details depend on the database. With MySQL, try for example

```
mysql -h localhost -u root -p
```

With Postgres, use the `psql` shell, and edit the `pg_hba.conf` file to allow `md5` or `password` access to the Postgres server from the network, including `localhost`.

9.4. RDB URLs and Schemata

Whatever database you use, in the end you need to provide CSS tools or users who want to access the data from other tools with the following configuration information:

- URL: Depending on the RDB system this will be a URL of the format

  ```
  jdbc:mysql://[host]:[port]/[database]
  ```

  for MySQL (default port is 3306),

  ```
  jdbc:postgresql://[host]:[port]/[database]
  ```

  for PostgreSQL (default port is 5432) or

  ```
  jdbc:oracle:thin:@//[host]:[port]/[service]
  ```

  for Oracle. CSS tools will use the start of that URL to detect the RDB dialect: MySQL, Oracle or PostgreSQL.
- User name: A user name known to the RDB. To end users, you will typically provide the name of a read-only account.
- Password: Associated password
- Schema: A prefix for RDB table names that might be needed by Oracle to access the tables.

To elaborate on the “Schema”, let's use an example. In the `ALARM` database, there is a `PV` table. When using MySQL, one can connect with a URL

```
jdbc:mysql://my.rdb.host/alarm
```

as any user and directly read from the `PV` table. Similar with PostgreSQL.

With Oracle, the URL usually addresses an Oracle service, not a database schema:

```
jdbc:oracle:thin:@//my.rdb.host:1521/prod
```

With Oracle, only the schema owner, that is a user with the same name as the database, can directly access the tables in a schema. All other users need to prefix the table name with the schema name, i.e. use `ALARM.PV` to read from the PV table.

To support all types of database dialects, the CSS tools allow configuration of a URL as well as a schema. For Oracle, you need to set that schema to the respective table prefix. **For MySQL and PostgreSQL, you typically define the schema as empty,** because the URL already includes the schema name in its `[database]` section.

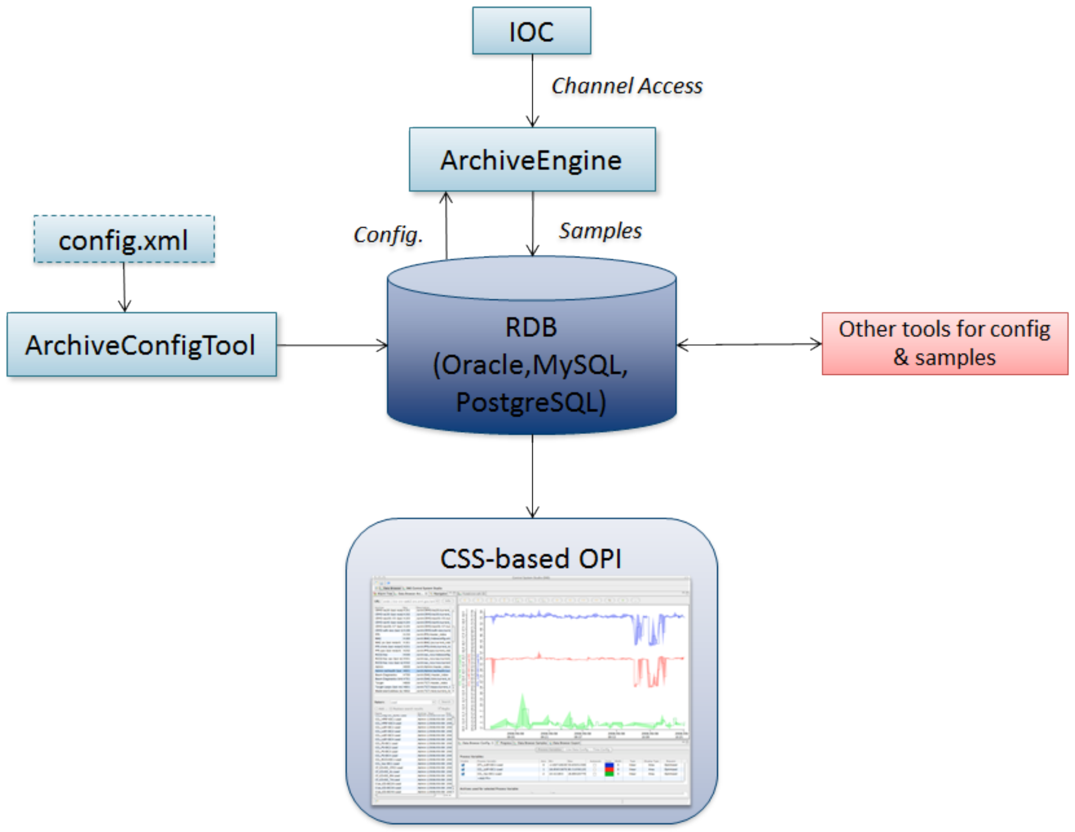
Chapter 10. Archive System

One part of CSS is the Archive System, specifically the “Best Ever Archive Toolset, yet (BEAUTY)” that was developed as a replacement for the Channel Archiver. An Archive Engine takes PV data samples from a front-end computer, for example from EPICS IOCs via Channel Access, and places them in some data storage, see Figure 10.1, “Archive System Overview”. Archive client programs then access historic data samples in that storage.

As described here, the storage is a Relational Database (RDB) like MySQL, Oracle or PostgreSQL. Both the historic data of PVs and the Archive Engine configuration are stored in the same relational database. The engine configuration can be imported from an XML file format into the database, or it can be exported from the database back into an XML file format for editing. The archive engine uses a pluggable implementation for its configuration and data storage as described in Chapter 30, *Archive Tools - org.csstudio.archive.engine and related*, so it is fundamentally possible to use the archive engine with different storage for the configuration and data, but in the following examples we concentrate on an RDB-based setup.

Typical setups will include more than one Archive Engine, for example one sample engine per subsystem. In principle, data providers other than archive engines can also write samples to the storage. The CSS Data Browser is a generic client program for looking at archived data, but fundamentally any program that has access to a relational database can be used to create reports. A typical application might be JSP-based web reports of data.

Figure 10.1. Archive System Overview



There are two integration points with the legacy Channel Archiver: The `ArchiveConfigTool` tool can import existing archive engine configuration files into the RDB because the XML file format is compatible with the Channel Archiver. The Data Browser is capable of reading data from the relational database as well as from the Channel Archiver's XML-RPC-based data server, thereby allowing nearly transparent access to both “old” and “new” data.

10.1. Relational Database Setup

Before using the archive tools, you need to create the required table structure in your RDB. Currently MySQL, Oracle and PostgreSQL are supported, see also Chapter 9, *Relational Database (RDB)*.

The commands for creating the RDB table structures are in files in the `dbd` sub-directory of the plugin `org.csstudio.archive.rdb`. Basic RDB administration skills will be required because you need to create the table structure by using one of these files, and will probably also need to create two accounts: One account for the archive engines that has write access to the tables, and another read-only account for archive clients like the CSS Data Browser to read archived data.

The RDB tables for the different database dialects are very similar with the exception of the `TIMESTAMP` used to store the time stamps of samples. While the Oracle time stamp data type already offers nanosecond detail, the MySQL and PostgreSQL data types of the same name only cover seconds. The MySQL tables therefore have an added `nanosecs` column for this purpose. There are a few more differences in the SQL dialects, but the Archive Engine and Data Browser auto-configure based on the database URL.

The setup for MySQL might be the easiest at least for development and testing, but it has limitations. All samples for all channels are written to one `sample` table. By default, MySQL table sizes are limited to 4GB (See MySQL `show table status` command, column “Max_data_length”). While this can be adjusted, I believe there is still a limit of 4G rows (=samples). Furthermore, while it will be almost trivial to enter something like

```
DELETE FROM sample WHERE smpl_time < ...
```

to delete older samples, this will either not free up any space or require an added `OPTIMIZE` rebuild, which takes a very long time.

For PostgreSQL, the table size limit at this time seems to be much higher at 32TB, with no additional row count limit. Performance of the `sample` table that holds archived samples is reduced by about 50% when adding constraints. The `dbd` file for PostgreSQL includes constraints to allow the RDB to assert referential data integrity, but if you trust the Archive Engine code to only write correct samples to the RDB, performance can be gained by disabling the `sample` table constraints.

One reason for using Oracle lies in its support for partitioning. While the sample `sample` appears as one table, it can be spread over several table partitions based on the sample time and channel name. Spreading by channel name might improve performance because several channels can be written in parallel to different disk locations. Partitioning by time allows quick removal of older samples. In addition, for Oracle the archive data readout implementation used by the Data Browser (plugin `org.csstudio.archivereader.rdb`) supports a stored procedure for server-side data reduction which is not available for MySQL.

Whatever database you use, in the end you need to provide all CSS archive tools with the following configuration information as elaborated in Section 9.4, “RDB URLs and Schemata”:

- URL
- User name
- Password

- Schema

For the Archive Engine and Archive Config Tool, you need to provide the user name and password of an RDB account that has write access to the archive tables. For the Data Browser or other tools that read data, a read-only account is sufficient. See Chapter 6, *Hierarchical Preferences* for details on how to provide these RDB settings for the archive plugins. The command-line tools will also accept these parameters on the command-line.

10.2. Building the Tools

The Archive Engine is the central sampling tool that reads values from PVs and writes them to the archive data storage. It is implemented as an Eclipse product. You will probably also want to build ArchiveConfigTool, the tool used to import engine configuration files into the relational database. They are defined in these product files:

```
org.csstudio.archive.engine/ArchiveEngine.product
org.csstudio.archive.config.rdb/ArchiveConfigTool.product
```

For first tests, you can run both tools from within the Eclipse IDE as described in Chapter 4, *Compiling, Running, Debugging CSS*, Section 4.3, “IDE Example”, but note that you will have to provide command-line arguments to them. After first tests are successful, you can export them from the IDE as described in the same section. Finally, you will need one of the CSS end-user products that includes the Data Browser to look at the archived data, but for now we concentrate on the tools needed to collect data.

10.3. Archive Engine Configuration

Each sample engine configuration identified by a name, for example “WaterSystem”. Inside the RDB the configuration is actually identified by a unique numeric ID, but most end user tools only see the name of the configuration.

Channel Groups

Each archive engine configuration is comprised of groups. An engine configuration has at least one group, maybe more, and channels are then added to these groups. Groups are not hierarchical: There are no sub-groups within groups, only one list of groups.

Groups are primarily used to organize the configuration. For example, a “WaterSystem” sample engine configuration might have groups “WestSector”, “MainBuilding” etc. to hold the channels for the respective section of the water system. Note that this arrangement of channels into groups is *not visible to end users of the data*! The separation of channels into groups inside the sample engine configuration is mostly meant for the engineers who maintain the sample engine configuration, grouping the channels by location along the machine, but associated front-end computer, or by functionality.

There is one functional aspects of groups: Archiving of all channels in a group can be enabled or disabled based on one channel in the group. When placing all channels of a power supply in a group, this feature can be used to suppress archiving of noise while the power supply is off by using a channel that indicates whether the power supply is on or off to enable the archive channel group.

Channels

A channel in the archive system is basically the data provided by one Process Variable. A channel is identified by its name, which has to be a valid PV name for the control system, a PV that you can also read with other control system tools. The samples stored for the channel include not only the value, for example a number, but also the time stamp, status/severity and meta data like engineering units and display ranges. The time stamp, status/severity and value are stored with each sample. The meta data is only stored once at startup of the archive engine because the original implementation for EPICS did not offer an efficient way to monitor for changes in the meta data.

When a channel sends a new value to the archive engine is somewhat outside of the control of the archive engine. The software on the front end computer controls this. For EPICS record, the `SCAN` field in combination with the `ADEL` field of analog records determines when a new value is sent to the archive engine.

The meta data for a channel is similarly controlled by the front end device that provides the data. For EPICS records, the `EGU`, `HOPR` and other fields have to be used to configure these.

Duplicate Channels

The RDB configuration allows for multiple sample engines. Each sample engine has one or more groups of channels, and each group has one or more channels. A channel, however, *can only be archived once*. It is illegal to list a channel in more than one group or under more than one sample engine.

10.4. Sample Modes

The archive engine supports several sample modes, i.e. ways in which it decides what samples should be written to the archive data store. As just mentioned in the section called “Channels”, the front-end computer decides which updates to send to the archive engine. In an ideal world, every such change would be meaningful and there were infinite resources (CPU power, disk space, network bandwidth) to store every change until eternity. In reality, it is often better to store fewer samples.

The archive engine supports the following sample modes by which it collects samples from a channel. Refer to the section called “Archive Config Tool” for an example of how these sample modes are specified in the XML format that can be used to configure an archive engine.

Monitored

In monitored mode, each received sample is written to the store. With a perfectly configured data source, for example an EPICS `ADEL` that only passes significant changes to the archive engine, this mode is ideal: Significant changes in value are written to the archive, while noise in the signal is suppressed to minimize wasted resources.

When configuring a monitored channel, the *estimated* time period between changes needs to be configured to allow the archive engine to reserve a suitable memory buffer where it stores received samples until they are written to the storage.

Monitored With Threshold

This mode is also monitored, but adding another value change threshold filter. Ideally, the front-end computer already performs the thresholding, so only significant changes are sent over the network to the archive engine. In some cases, however, this is not possible, and for those cases the archive engine itself can check for changes in the value, writing only samples that differ from the last written sample by at least some configurable margin.

As with plain monitored channels, the *estimated* time period between changes needs to be configured.

Scanned/Sampled

In scanned mode, the archive engine still receives each update from the data source, but it only writes the most recent sample at periodic times, for example once every 5 minutes.

For a scanned channel you configure the period at which the archive engine should check the channel for its current value.

This mode is a compromise. If a channel has no significant change for hours, why should the uninteresting changes fill disk space every 5 minutes? On the other hand, if an important even happens that produces a brief “blip” in the data, the archived data is likely to miss it when only storing a value every 5 minutes.

This mode was created for channels which do not have a good dead-band configuration, where using the monitored mode would add too many samples to the archive. Periodic sampling is clearly imperfect, but sometimes a workable compromise.

Writing Samples to Storage

Samples obtained by the various samples modes are **not** immediately written to storage, for example the RDB, because writing each individual sample right away would be too slow. Instead, samples are initially kept in memory, then written to storage in bulk. By default, this write period is 30 seconds.

The period configured for scanned channels or the estimated change period for monitored channels is used to allocate the in-memory buffer that the engine uses to collect samples between writes. The in-memory buffer is a ring buffer that is written each write period. If a monitored channel sends many more samples than configured via the estimated update period of a channel, the archive engine sample buffer for that channel will overrun older in-memory samples for that channel. The archive engine actually uses a buffer reserve to allocate a slightly bigger in-memory buffer to avoid such overruns:

```
buffer_size = write_period / scan_period * buffer_reserve
```

The default buffer reserve is 2. With the default write period of 30 seconds, a channel with an estimated change period of 2 seconds would thus expect to need to buffer 15 samples between writes to storage, but the the actual buffer size would be 30 to prevent ring buffer overwrites during times where writing to storage is slightly delayed, or a few more samples are received than originally expected. The in-memory buffer still has a fixed size, it will not grow when more samples are received to keep a constant memory footprint for the archive engine.

When writing accumulated samples for all channels to storage, i.e. by default every 30 seconds, the samples are written in batches. For RDB-based storage, the JDBC statements are batched to reduce the number of individual commits to the RDB. The default batch size is 500.

Time Stamp Checks

When viewing archived data, the time stamps of historic samples are obviously quite important. The Archive Engine simply receives time-stamped data from front end computers and has no way to determine if those time stamps are correct. It enforces, however, a few basic rules:

- Zero time stamps: Time stamps with zero seconds, for example EPICS time stamps with zero seconds since the EPICS epoch of 1970, are ignored.
- Time stamps that go back-in-time, i.e. time stamps that are before samples that have already been inserted into the archive, are ignored.
- Futuristic time stamps, that is time stamps that are too far ahead of the clock of the host that is executing the archive engine, are ignored. By default, this ignored future is 1 day.

10.5. Editing the Configuration

The configuration for all archive engines resides in the RDB, which allows you to modify it in various ways. Note, however, that running archive engines are not notified of configuration changes in the RDB because there is currently no convenient mechanism for them to learn about such changes. You *must manually re-start all affected archive engines* after modifying their configuration!

SQL Manipulation

It is possible to modify an archive engine configuration via direct SQL manipulation, for example from an SQL shell:

```
SELECT * FROM smpl_eng;
INSERT INTO smpl_eng(name, descr, url)
  VALUES ('demo', 'Example Engine',
          'http://somehost:4812');
```

This clearly requires some familiarity with the RDB table layout, see Section 10.1, “Relational Database Setup”. For operations like renaming a channel or bulk changes this can be the most convenient procedure.

Archive Config Tool

The `ArchiveConfigTool` can export existing archive engine configurations from the RDB into an XML file format, or import such XML files into the RDB. The XML file format is compatible with the one used by the Channel Archiver, allowing the import of existing archive engine configurations.

The `xml` directory in the plugin `org.csstudio.archive.config.rdb` contains a commented example configuration file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Example for XML configuration file syntax -->
<engineconfig>
  <!-- Engine can have one or more groups
       Each group has a name and one or more channels
    -->
  <group>
    <name>NorthSectorVacuum</name>
      <!-- Each channel has a name and
           a sample period (or expected change period).
           The period is either in seconds or in HH:MM:SS format.
           It is either using the 'monitor' or 'scan' sample mode.
       -->
      <channel>
         <name>NSV:P1</name>
         <period>1.0</period><monitor/>
      </channel>

      <!-- Example for a monitor with engine-enforced
           value change threshold of 2.5
        -->
      <channel>
         <name>NSV:P2</name>
         <period>1.0</period>
         <monitor>2.5</monitor>
      </channel>

      <!-- Channel that is scanned every 10 minutes-->
      <channel>
         <name>NSV:Enable</name>
         <period>00:10:00</period><scan/>
         <enable/>
      </channel>

      <!-- Channel that enables sampling of this group -->
      <channel>
         <name>NSV:Enable</name>
         <period>1.0</period><monitor/>
         <enable/>
      </channel>
  </group>

  <!-- Other Group -->
  <group>
    <name>SouthSectorVacuum</name>
      <channel>
         <name>SSV:P1</name>
         <period>1.0</period><monitor/>
      </channel>
  </group>
</engineconfig>
```

The ArchiveConfigTool offers command-line help similar to this:

```
-help                      : show help
-engine my_engine          : Engine Name
-config my_config.xml      : XML Engine config file
-export                    : export configuration as XML
-import                    : import configuration from XML
-delete_config             : Delete existing engine config
-description 'My Engine' : Engine Description
-host my.host.org          : Engine Host
-port 4812                 : Engine Port
-replace_engine            : Replace existing engine config, or stop?
-steal_channels            : Steal channels from other engine
-rdb_url jdbc:...          : RDB URL
-rdb_user user             : RDB User
-rdb_password password     : RDB Password
-rdb_schema schema         : RDB schema (table prefix), ending in '.'
```

To export an existing engine configuration into an XML file, use

```
ArchiveConfigTool -engine my_engine -config my_config.xml -export
```

In addition, the RDB connection parameters might have to be supplied unless they are built into the tool or provided via a `-pluginCustomization` argument.

To import a configuration from an XML file into the RDB, use

```
ArchiveConfigTool -engine my_engine -config my_config.xml -import\
-host my.host.org -port 4812
```

The host name specifies the host on which the engine is supposed to execute, and the port number under which is runs its web server. The configuration file could be the edited result of a previous export, or have been created by other means.

The archive config tool is cautious about disturbing existing configurations. By default it will stop when there is already a configuration in the RDB for the same sample engine name unless the `-replace_engine` option is provided, in which case an existing configuration for that engine name will be deleted before importing the XML file. Similarly, the config tool will ignore channel names that are already handled by a different engine configuration, unless the `-steal_channels` option is provided to instruct the tool to move such channels from the archive engine that previously handled the channels.

10.6. Running Archive Engines

The `ArchiveEngine` is a headless RCP application that reads a sample engine configuration, connects to the control system channels listed in the configuration, and writes received samples to the archive data store.

It supports these command-line arguments:

```
-help                         : Display Help
-port 4812                    : HTTP server port
-engine demo_engine           : Engine config name
-data /home/fred/Workspace    : Eclipse workspace location
-pluginCustomization /path/to/mysettings.ini: Eclipse plugin defaults
```

The `-pluginCustomization` parameter can be used to provide settings for the RDB connection, to configure the logging, and to provide settings for access to the control system, for example EPICS Channel Access network preferences.

The `-engine` parameter selects the sample engine configuration. In principle, that engine configuration already includes the URL of the engine web server, but an additional `-port` parameter is required for two reasons: First, this allows the engine to start a web server which can be used to monitor engine operation as soon as possible, for example while the engine it trying to connect to an RDB. Secondly, the engine will compare the provided port number with the port number of the URL in its configuration. This is means as a basic constistency check that helps avoid running archive engines with the wrong configuration.

Engine Web Server

Each sample engine has a built-in web server for status information and basic remote control of the engine. When starting the engine on a host, the port number for this HTTPD must be provided. The sample engine URL configured in the RDB should match the format

```
http://<host>:<port>/main
```

The engine will compare the port number from the URL with the port number provided as a command-line argument.

The engine web server provides several web pages, mostly linked from the `.../main` URL, that allow you to see:

- Is the engine running? Since when?
- Are all channels connected? Which are disconnected?
- What is the last data that a channel has received? What is the last sample that was written to the storage?

In case of problems, the last item is usually helpful to determine: Does the front end computer send correct time stamps? Does the data change to qualify for writing a new sample to the storage?

Note that the engine only serves a blank page at its root URL. For example, accessing `http://localhost:4813` will result in an empty page. You have to start browsing at `http://localhost:4813/main`. Starting at `.../main`, one can drill down to the status of groups and individual channels.

A few engine web pages are *not* accessible by following web browser links because they affect the engine operation. This is meant to prevent a web-crawling program to accidentally stop the engine.

Other Engine URLs

`http://<host>:<port>/stop`	Invoke this URL to stop the engine gracefully, i.e. to ask the engine to write a final `Archive_Off` sample to each channel, then quit.
`http://<host>:<port>/restart`	Invoke this URL to trigger a running restart of the engine. The engine will stop sampling, read its configuration, then start again. Invoking this URL is required after changes to the configuration of an archive engine.
`http://<host>:<port>/reset`	Invoke this URL to reset engine statistics, for example the average write time displayed on the main page of the engine.
`http://<host>:<port>/environment`	Invoke this URL to display engine environment settings which might be useful when trying to debug a problem.

10.7. Web-Based Archive Monitor and Editor

At the SNS, a JSP-based collection of reports can display graphs of the archive system performance, for example: Which archive engine wrote how many samples to the archive over the last hour? It also includes a web-based editor for the alarm system configuration.

This reporting package, however, is currently part of a bigger, more SNS-specific reporting package. Contact Kay Kasemir if you are interested in collaborating on a more portable version of these reports.

10.8. Viewing Data in CSS

The CSS Data Browser uses the plugin `org.csstudio.archive.reader` to access archived data in general. The plugin `org.csstudio.archive.reader.rdb` provides access to data stored in an RDB written by the archive engine described in this chapter, i.e. it implements reading from URLs of the form `jdbc:....`.

For a successful retrieval, you need the following:

- Include `org.csstudio.archive.reader.rdb` in your CSS product.
- Configure it with the correct (read-only) RDB user and password.
- Configure the Data browser to use URLs like `jdbc:mysql://localhost/archive` to read from the archive.

Chapter 11. Java Message Server

CSS uses a Java Message Server (JMS) for several purposes. Log messages from CSS applications in general can optionally be sent to JMS, see Chapter 23, *Logging - org.csstudio.logging*. The alarm system uses JMS for the communication between the alarm server and clients.

11.1. Apache ActiveMQ Server

CSS uses Apache ActiveMQ, a free and open-source implementation of JMS available from http://activemq.apache.org.

Fundamentally, ActiveMQ is a portable Java application, but the scripts to start it are slightly different for Windows and other operating systems, so assert that you download the version suitable for your computer. After downloading it, the ActiveMQ server can be started like this:

```
cd [activemq_install_dir]
bin/activemq
```

At least that used to be the case. With recent versions, an additional argument `start` to start respectively `stop` to stop the server seems to be required. As a very basic test on Unix and OS X, you can check if the server is listening on port 61616:

```
netstat -an | fgrep 61616
```

The page http://sourceforge.net/apps/trac/cs-studio/wiki/JavaMessageServer has more details on how to adjust the default JMS configuration. For example, you probably want to disable parts of JMS that are not required for operation with CSS.

11.2. Client (CSS) Configuration

To use JMS for logging or the alarm system, CSS needs the URL of your JMS instance. It should be of the form

```
failover:(tcp://your_jms_host:61616)
```

“Failover” with a single server instructs the client to automatically re-connect after network problems. Actual failover between multiple JMS instances is also possible, but for details you need to refer to the Apache ActiveMQ documentation.

11.3. Testing your Setup

As a basic test of your JMS setup, you can enable JMS logging in your CSS product, for example by adding these lines to your plugin customization file (see Chapter 6, *Hierarchical Preferences*):

```
org.csstudio.logging/console_level=FINE
org.csstudio.logging/jms_level=FINE
org.csstudio.logging/jms_url=failover:(tcp://your_jms_host:61616)
```

As a result, CSS should send several log messages to JMS. You can verify this by starting the JMS Monitor, see Chapter 31, *JMS Monitor - org.csstudio.debugging.jmsmonitor*, to view messages for the `LOG` Topic.

While you should probably disable this in a production setup, the default configuration of ActiveMQ includes a web browser interface. It is accessible at http://localhost:8161/admin on the host where JMS is running. It can display who is connected to JMS and what topic each connection is reading or writing.

11.4. Message Types

All JMS messages used by CSS are of the `MapMessage` type. Fundamentally, this allows messages with arbitrarily named string properties for content. In reality, however, messages are most useful when the involved applications understand the messages. For example, every message shall have a `TYPE` property. CSS log messages as sent by the plugin `org.csstudio.logging` (see Chapter 23, *Logging - org.csstudio.logging*) have the following properties:

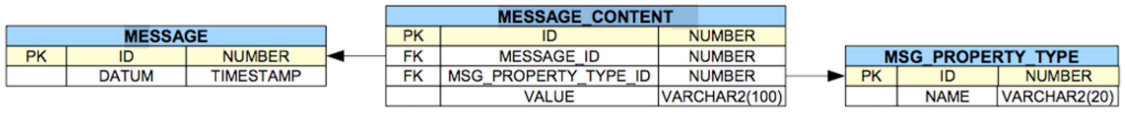
- `TYPE`: Set to “log” to identify as log message.
- `TEXT`: The actual log message.
- `SEVERITY`: Log level, for example “SEVERE” or “FINE”. The exact severity may depend on the underlying logging system.
- `CREATETIME`: Time when message was created. Format must be `yyyy-MM-dd HH:mm:ss.SSS`
- `CLASS`: Name of (Java) class were message was created.
- `NAME`: Name of Java method
- `APPLICATION_ID`: Application name like “CSS” or “AlarmServer”
- `HOST`: Host name running the application.
- `USER`: Name of user who was running the application.

The content of messages exchanged within the alarm system or from for example a tool that logs write actions from an operator interface should try to use the same properties as much as possible.

11.5. JMS logging to RDB

The original RDB schema for logging JMS messages as shown in Figure 11.1, “Message RDB Schema”was developed for the DESY version of CSS.

Figure 11.1. Message RDB Schema



The column “DATUM” (German for date) holds the time when a message was written to the RDB. All other message properties like `TEXT` and `CREATETIME` are written to the MESSAGE_CONTENT and MSG_PROPERTY_TYPE tables. This schema is very compact and generic. On the other hand, it is operationally often useful to search for all messages of TYPE=log or with a certain SEVERITY. Such searches are relatively slow in the original schema because they require nested lookups in the MESSAGE_CONTENT table.

The SNS version of CSS therefore added commonly used message properties directly to the MESSAGE table. Some tools like the message viewer from `org.csstudio.alarm.beast.msghist` automatically determine which message properties are in the main MESSAGE table, and which are in the MESSAGE_CONTENT table. Other tools like the ones used to write messages from JMS to the RDB are site-specific, see Chapter 33, *RDB Logging - org.cstudio.sns.jms2rdb*

11.6. Viewing the Message History

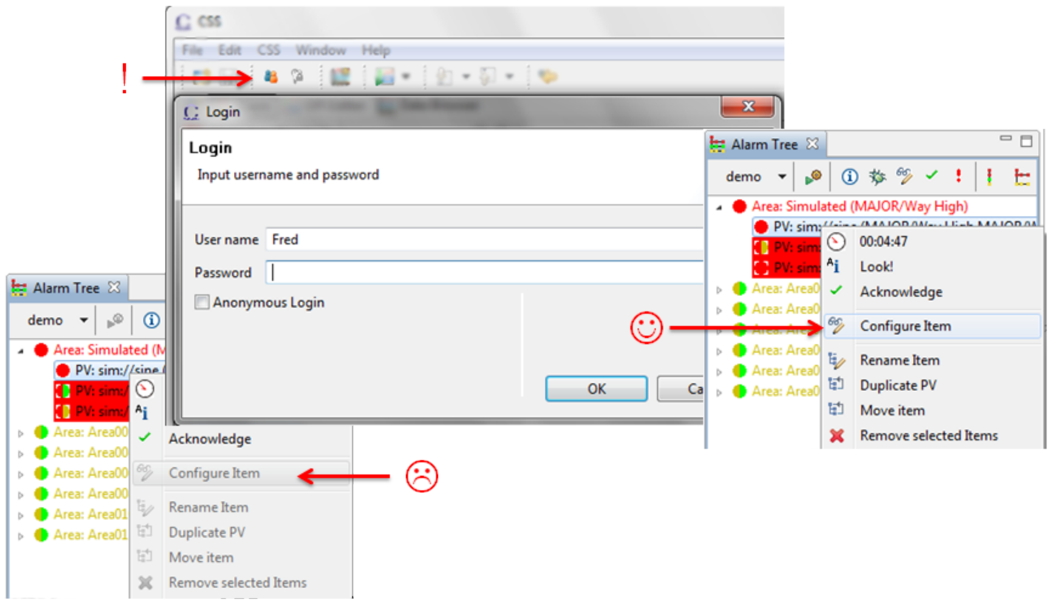
CSS includes a generic message history browser as part of the alarm system, see Chapter 29, *Message History Browser - org.cstudio.alarm.beast.msghist*. Once the messages are logged to the relational database, it is of course possible to create various tools to create customized reports, for example based on JSP technology for web reports.

Chapter 12. Authentication and Authorization

12.1. Overview

Access to some components of CSS is restricted. Changes to the alarm system configuration for example are only permitted to authorized users. User interface elements like the context menu entries to configure or remove some part of the alarm configuration are only accessible after the user has logged in *and* that user is indeed authorized to manipulate the alarm configuration.

Figure 12.1. Authenticate to change alarm configuration



Authentication is the act of confirming a user's identity, typically by prompting for a *user name* and *password*, and trusting that the user is who he claims to be if the password checks out OK.

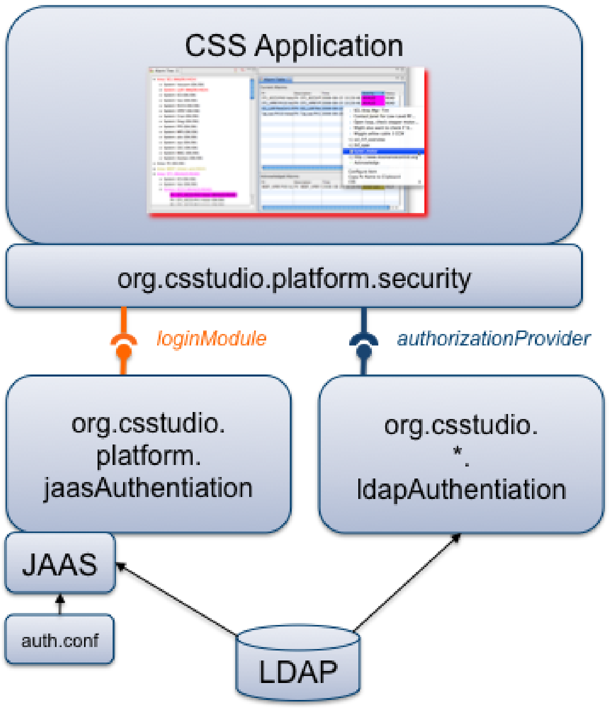
Authorization is the process of determining if an *authenticated* user is allowed to perform a certain operation, typically by consulting some type of database that lists the permissions of all the known users.

CSS has a security API for authentication and authorization (Auth & Auth) defined by the plugins `org.csstudio.auth` and `org.csstudio.auth.ui`, and there are several plugins that provide implementations.

Most CSS products will have to include `org.csstudio.auth` because other plugins in the product use its API. Your product should also include `org.csstudio.auth.ui`. As soon as you include one of the implementations described in the following sections, the `auth.ui` plugin will then offer login buttons in the toolbar and corresponding menu entries. In addition to including the `auth.ui` plugin, your product needs to register a Login Callback Handler, something that provides a user name and password by for example displaying a log-in dialog. An example for registering such a Login Callback Handler can be found in `org.csstudio.utility.product.Workbench`.

A certain understanding of these mechanisms is required even if you want to start out by providing all users with full access to all features, because missing Auth & Auth support can result in restricted access for all users. The overall CSS Auth & Auth design is probably more complicated than it needs to be, but in here we only try to describe it. As a minimum, you might have to configure your CSS product to use the “dummy” authentication and authorization.

Figure 12.2. CSS Extension Points for Auth & Auth



12.2. Authentication: org.csstudio.platform.loginModule

This CSS extension point allows plugins to provide authentication. CSS expects *exactly one* implementation. If multiple implementations are found, the result is the same as having none: Nobody can authenticate. Authentication may be invoked when starting CSS as part of a “log in” dialog. It is later also available via the menu item `File`, `Switch User`.

org.csstudio.platform.jaasAuthentication

This plugin implements authentication based on JAAS, the Java Authentication and Authorization Service. JAAS in turn is highly configurable to use for example LDAP. The plugin contains a JAAS configuration file `conf/auth.conf`. The file contains the following examples to get you started, but you might have to consult the JAAS documentation for details:

- Dummy: Accepts every user name and password except for the user name “fail”. Useful for testing or when you don't really want to support authentication.
- Kerberos
- LDAP for a few LDAP servers

Since the `auth.conf` file can list many possible configurations, the jaasAuthentication plugin uses a preference setting to determine which configuration to use. You should set this preference in the `plugin_customization.ini` file of your CSS product:

```
# Select 'Dummy' JAAS Authentication
org.csstudio.platform.jaasAuthentication/jaas_config_source=File
org.csstudio.platform.jaasAuthentication/jaas_config_file_entry=Dummy
```

To add your authentication method, you could extend the `auth.conf` file and then select your entry via the `jaasconfig` preference.

Alternatively, you can use the preference setting `jaas_config_source=PreferencePage` and then provide what you would have added to the file in a `jaas_prefs_config` preference variable. For examples see `preferences.ini` in the jaasAuthentication plugin.

org.csstudio.platform.jaasAuthentication.ui

This plugin allows setting of jaasconfig preferences from the Eclipse preference GUI.

JAAS LDAP Authentication via JndiLoginModule

To use LDAP authentication with the standard JAAS `JndiLoginModule`, your LDAP server must provide user and password information in `posixAccount` entries as they are also used for Linux accounts.

```
# LDAP user/password example suitable for JAAS
# com.sun.security.auth.module.JndiLoginModule
dn: uid=fred,ou=People,dc=test,dc=ics
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: posixAccount
objectClass: top
uid: fred
userPassword: {CRYPT}xxxxxxxxxx
cn: Fred Testuser
sn: Fred
homeDirectory: /home/fred
uidNumber: 42
gidNumber: 42
```

Fundamentally, the user name and password entered by the user must match the `uid:` and `userPassword:` in LDAP. Note that the userPassword **must be in `{CRYPT}` format** because the JAAS JndiLoginModule only handles the `{CRYPT}` format and cannot read passwords stored in a different format.

The JndiLoginModule also requires `uidNumber:` and `guiNumber:` entries as part of the posixAccount object class, even though CSS will not use that information. For full details refer to the javadoc of the JAAS JndiLoginModule.

JAAS LDAP Authentication via LDAPBindLoginModule

The standard JAAS `JndiLoginModule` requires read access to the password information in LDAP. If your LDAP server administrator does not allow this for security concerns, you can select the custom JAAS login module

```
org.csstudio.platform.internal.jaasauthentication.LDAPBindLoginModule
```

This module authenticates to LDAP by using a simple bind operation. The entries on the LDAP server look as in the `JndiLoginModule` JndiLoginModule case, except that the only thing which really matters is the `dn:` and `userPassword:` information. The LDAPBindModule will try to bind to a DN, and the LDAP server checks the password. The login module will not attempt to read anything from LDAP, so the remaining entries are irrelevant.

Depending on your LDAP server, beware that it may only handle the bind for passwords in a certain format. Specifically, it may not allow a bind for entries with `{crypt}` formatted passwords, which clashes with the previous case.

JAAS Kerberos Authentication

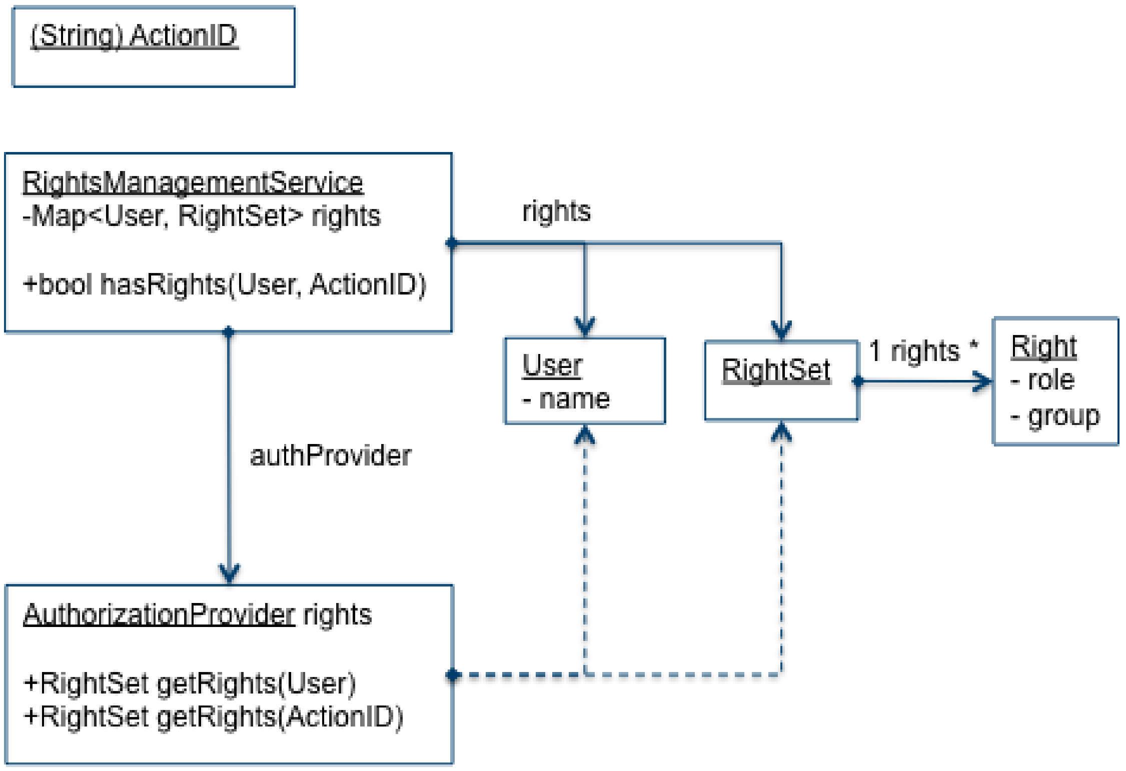
The `conf/auth.conf` file in the jaasAuthentication plugin contains an example for using the JAAS Krb5LoginModule, which in turn uses Kerberos. Additional environment variables may be needed, refer to the JAAS documentation.

12.3. Authorization: org.csstudio.platform.authorizationProvider

This extension point allows plugins to provide authorization. The data model is based on Users who have Rights, and Actions that require Rights. Rights are described via Roles and Groups:

- `User`: Name of an authenticated user, for example “Fred”.
- `Action ID`: Identifier for a certain action, for example “alarm_ack” for the acknowledgement of alarms. Also called Authorization ID.
- `Role`: A site-specific role that a user might have, for example “ADMIN”, “OPERATOR” or “GUEST”.
- `Group`: A site-specific group like “VACUUM” or “CRYO”.
- `Right`: The combination of role and group. User “Fred” might be an “ADMIN” in the “CRYO” group but only a “GUEST” in the “VACUUM” group.

Figure 12.3. CSS Authorization Data Model. Applications actually use the `SecurityFacade` API which internally calls the `RightsManagementService`.


(String) ActionID
RightsManagementService
-Map<User, RightSet> rights
+bool hasRights(User, ActionID)
rights
User
- name
RightSet
1 rights *
Right
- role
- group
authProvider
AuthorizationProvider rights
+RightSet getRights(User)
+RightSet getRights(ActionID)

org.csstudio.sns.dummyAuthorization

This plugin always authorizes everybody to do anything. Note that it is still required to be *authenticated*, because authorization is only checked for authenticated users. Non-authenticated users have no rights.

org.csstudio.sns.ldapAuthorization

This plugin stores the Authorization info in LDAP. The rights assigned to each user and the rights required to execute a certain action are in two separate sections under a common base DN. The following examples use a base DN of `ou=CSSAuthorization,dc=test,dc=ics`.

Before entering CSS authorization information, you need to add the following schema definitions to your LDAP server:

```
# css.schema
# Authorization related scheme elements
# Based on information from http://css.desy.de

objectclass ( 1.3.6.1.4.1.341.999.2 NAME 'cssRole'
  DESC 'Role for CSS authorization: cn is role, ou is group'
  SUP top STRUCTURAL
  MUST cn
  MAY ( memberUid $ description )  )
```

```
attributetype ( 1.3.6.1.4.1.341.999.1.3 NAME 'auid'
  DESC 'name of authorize ID'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256} )

attributetype ( 1.3.6.1.4.1.341.999.1.2 NAME 'cssGroupRole'
  DESC 'group and role, it must be in a format of (group,role)'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256} )

attributetype ( 1.3.6.1.4.1.341.999.1.1 NAME 'cssActionPath'
  DESC 'the full path for an action or control in css'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256}  )

objectclass ( 1.3.6.1.4.1.341.999.1 NAME 'cssAuthorizeID'
  DESC 'an authorized ID'
  SUP top STRUCTURAL
  MUST auid
  MAY ( cssGroupRole $ cssActionPath $ description ) )
```

With OpenLDAP you would add the following to slapd.conf:

```
include  /etc/openldap/schema/css.schema
```

With OpenLDAP it might also be useful to control access to the authorization info with an entry similar to this in `slapd.conf`:

```
# Define which users can write to CSS Authorization
Access to dn.subtree="ou=CSSAuthorization,dc=test,dc=ics"
  by dn.exact="uid=alarm_config_user1,ou=Users,dc=test,dc=ics" write
  by dn.exact="uid=alarm_config_user2,ou=Users,dc=test,dc=ics" write
  by * read
```

Rights assigned to a user

The `CSSGroupRole` sub-tree lists all available rights and assigns users to them in entries based on the `cssRole` schema.

Example:

```
dn: cn=developer,ou=alarm,ou=CSSGroupRole,ou=CSSAuthorization,
   dc=test,dc=ics
objectClass: cssRole
cn: developer
memberUid: Fred
memberUid: Jane

dn: cn=operator,ou=alarm,ou=CSSGroupRole,ou=CSSAuthorization,
   dc=test,dc=ics
objectClass: cssRole
cn: operator
memberUid: operator
```

This example defines the following user rights:

- Right “developer”/“alarm”: Users authenticated as “Fred” or “Jane” have the “developer” role in the “alarm” group.
- Right “operator”/“alarm”: Users authenticated as “operator” have the “operator” role in the “alarm” group.

Rights required by an Action

The CSSAuthorizeID sub-tree lists the Rights required by restricted actions with entries based on the cssAuthorizeID schema.

Example:

```
dn: auid=alarm_ack,ou=AlarmSystem,ou=CSSAuthorizeID,
   ou=CSSAuthorization,dc=test,dc=ics
objectClass: cssAuthorizeID
auid: alarm_ack
cssGroupRole: (alarm, developer)
cssGroupRole: (alarm, operator)

dn: auid=alarm_config,ou=AlarmSystem,ou=CSSAuthorizeID,
   ou=CSSAuthorization,dc=test,dc=ics
objectClass: cssAuthorizeID
auid: alarm_config
cssGroupRole: (alarm, administrator)
cssGroupRole: (alarm, developer)
```

This example defines the following Action IDs:

- alarm_ack: The CSS alarm system uses this action ID for alarm acknowledgement. Users who want to acknowledge alarms need either alarm/developer or alarm/operator rights.
- alarm_config: The CSS alarm system uses this ID for alarm configuration actions. Users who want to change the alarm system configuration need the alarm/developer right.

Note how the example entries were listed under ou=AlarmSystem, one level below the CSSAuthorizeID sub-tree. This is purely for administrator convenience to allow grouping of related authorization requirements, for example to put all alarm system related Action IDs into a sub-tree. The authorization code will search all sub-trees of CSSAuthorizeID for entries in the cssAuthorizeID objectClass.

By combining the authorization configuration listed above, you get the following:

- A user authenticated as “operator”, which is likely a shared account, can acknowledge alarms.
- Individuals authenticated as “Fred” or “Jane” can acknowledge and configure alarms.

org.csstudio.platform.ldapAuthorization

Also based on LDAP, but with a different LDAP schema.

Chapter 13. Alarm System

This chapter introduces the CSS alarm system, specifically the “Best Ever Alarm System Toolkit” or BEAST. It was developed based on experience with the original EPICS alarm handler ALH combined with ideas from the book *Alarm Management: Seven Effective Methods for Optimum Performance* by B. Hollifield and E. Habibi, published by ISA in 2007.

13.1. Motivation

The primary goal of the alarm system is simple:

Effectively help operators take the correct action at the correct time.

This is easier said than done! One way to accomplish this is to adopt the following guiding principles for an alarm handling philosophy:

- Alarms are presented with guidance, related displays.

  Lacking these, how can operators effectively react to an alarm?
- Manageable alarm rate

  The number of alarms should stay below about 150 per day. If the alarm rate gets much higher, the alarm system will no longer help operators take correct actions but instead overload them with information.
- Operators will respond to every alarm.

  This is a corollary to the previous item: Assuming a manageable alarm rate, operators are expected to respond to each alarm.

To implement such an alarm handling philosophy, a lot of effort needs to be put into defining useful alarms. Each alarm must have guidance, clearly describing what operators need to *do* in response to an alarm. There should not be any “bogus” alarms from systems that undergo maintenance, or that are currently irrelevant for the operation of the machine. Alarms must not “chatter”, causing unnecessary noise by going in and out of alarm. Operators need convenient access to the currently active alarms and their associated information. The alarm system tools described in the following try to help:

- Alarms are presented in user-selectable ways: Table of current alarms, sorted as desired, but also a tree-view, displaying all or only active alarms. In principle, additional views can be added.
- Each alarm is presented with additional information like guidance on how to handle the alarm. There are links to related operator screens, web pages, and other CSS tools.
- The alarm system configuration can be edited from within the alarm system user interface.

13.2. Alarm Trigger PVs

The alarm system handles the display of alarms and associated information. The triggers of alarms are simply Process Variables (PVs) in the control system, outside of the alarm system per se. The alarm system monitors such PVs, and a non-normal severity will trigger an alarm.

In some cases, existing control system PVs that already have suitable limits can be used as trigger PVs for the alarm system. In other cases, new PVs may have to be created to serve as alarm trigger PVs. For EPICS, this can be done on IOCs associated with the respective subsystem of the machine, or in soft IOCs that are purely meant to serve alarm trigger PVs. In any case, alarm trigger PVs must generate a non-normal severity like “MINOR” or “MAJOR” to trigger an alarm.

13.3. Alarm System Behavior

The alarm system will typically *latch* alarms. This means that the alarm trigger PV can return to OK, later re-enter the alarm state and so on, but the alarm system will only react when a PV enters an alarm state for the first time. Subsequently, the alarm system user interface will display the current state of the PV, but it will not trigger a new alarm nor issue another annunciation. This is meant to reduce noise from the alarm system.

Once operators are able to react to the original alarm, operators *acknowledge* the alarm, and the alarm trigger PV returns to normal, the alarm will *clear*. This is a typical time line:

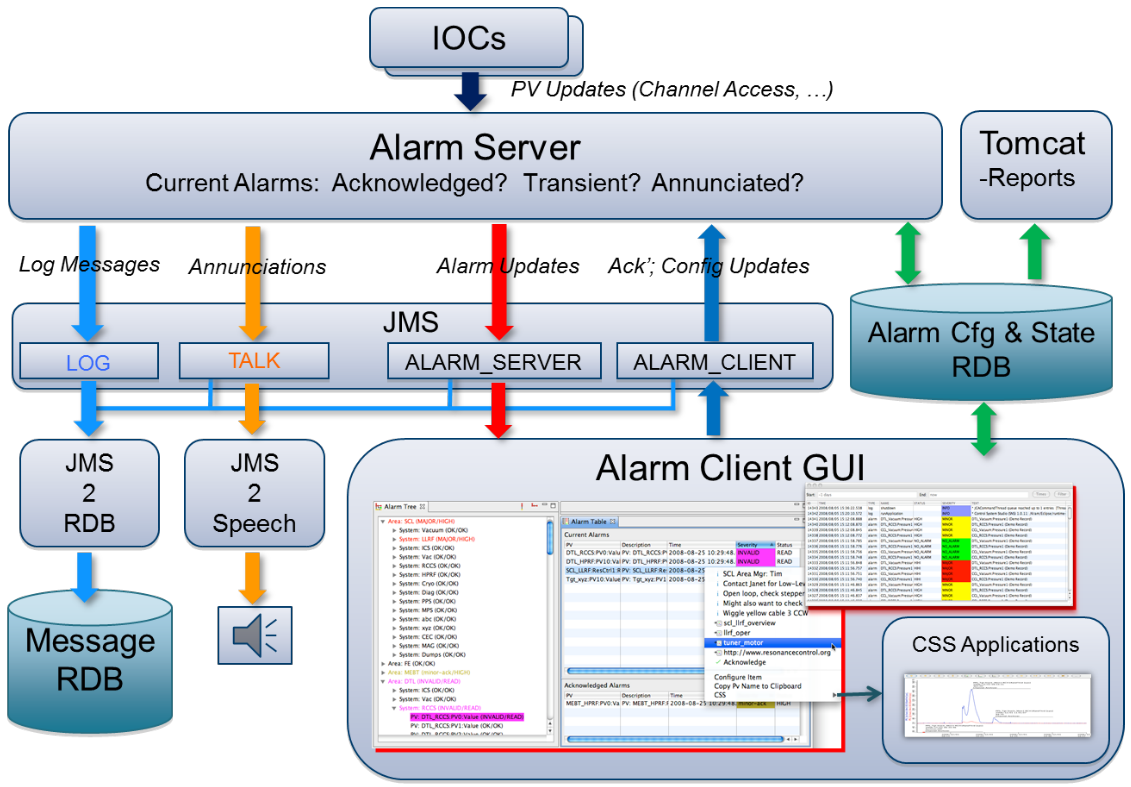
1. Alarm trigger PV X enters an alarm state, for example with “MINOR” severity.
2. The alarm system will indicate a new MINOR alarm for PV X. If configured to do so, it will also perform a voice annunciation of the alarm.
3. In a perfect world, the PV will stay in alarm until the underlying problem has been handled. In reality, the PV may briefly leave the alarm state, for example as a result of noise in a sensor. The alarm system will indicate this, but it still shows that there was a MINOR alarm with the time when it originally happened. As the PV re-enters a MINOR severity, there is no new alarm annunciation because this is considered part of the original alarm that has not been handled, yet.
4. Hopefully, operators can address the underlying issue soon, the alarm trigger PV returns to a normal severity, and operators acknowledge the alarm.
5. The alarm clears. It does not matter if the PV returns to normal, then operators acknowledge the alarm, or operators first acknowledge the alarm, then the PV returns to normal. The alarm will clear as soon as both conditions are met: PV returns to normal, alarm has been acknowledged.

Alarms may be configured not to latch, which means there is no need to acknowledge them. As soon as the alarm trigger PV returns to normal, the alarm will also clear. This mode of operation is primarily meant for alarm trigger PVs that already latch within the control system. An example would be an alarm trigger PV related to the “trip” condition of a device that requires a manual reset by operators to clear. That manual reset of the device is functionally equivalent to acknowledging the alarm, because an operator has noticed the condition and taken corrective action. The alarm trigger PV is unlikely to chatter because it latches until the device is reset. Requiring operators to acknowledge the alarm in addition to resetting the alarm trigger PV would cause unnecessary work.

One might be tempted to configure frequent alarms as non-latching to lessen the load on operators who get tired of acknowledging such a nuisance alarm. Overall, however, it is better to re-engineer such alarms. Add a filter to avoid alarms from brief occurrences of the symptom. Find a way to alarm on a precursor of the frequent alarm, allowing operators to avoid running the machine in a state that will later cause frequent occurrences of the nuisance alarm. Ideally, fix the underlying hardware or software to avoid the nuisance alarm altogether.

13.4. Technical Overview

Figure 13.1. Alarm System Overview



Fundamental to the operation of the alarm system is the `AlarmServer`. It reads the alarm system configuration from a relational database and monitors alarm system trigger PVs. Whenever a new alarm occurs, it informs alarm system user interfaces (GUIs) via JMS. For alarms that should be annunciated, it will inform annunciation tools via a designated JMS topic.

The Alarm Server performs the latching alarm behavior described in the previous section. As the severity of an alarm trigger PV changes, the server updates the GUIs via JMS about the current state of the PV, but it maintains the latched alarm state.

When operators acknowledge an alarm in the GUI, they are sending an acknowledge request via JMS to the Alarm Server. The server will in turn reply with the updated, eventually the cleared alarm state.

In addition to sending state updates via JMS, the Alarm Server also updates the alarm state in the relational database. This is done to allow newly started alarm GUI clients to obtain the initial state of all alarms. The persisted alarm state in the RDB also allows for Alarm Server restarts without loosing the alarm state.

Alarm GUI clients can change the alarm system configuration in the RDB. They use JMS to notify the Alarm Server and other GUI clients about the change, who then in turn read the updated configuration from the RDB.

As described, all alarm traffic passes through JMS: Alarm state updates, annunciations, acknowledgements, configuration changes. A generic JMS-to-RDB logger can thus capture all alarm traffic into an RDB for later review and anaysis, for example: What happened when? Which alarm was most frequent?

Alarm Tree “Root”

The RDB can hold more than one alarm system configuration. Each alarm configuration is identified by the name of its “Root” element, for example CCR for a Central Control Room configuration.

The Alarm Server and associated user interface clients must use the same alarm tree root. They will construct the names of JMS topics used to communicate by adding suffixes to the alarm tree root name. For example, with an alarm tree root of CCR, the server and clients will use the following JMS topics:

- CCR_SERVER: Used by the server to send alarm state updates to clients.
- CCR_CLIENT: Used by clients to send acknowledgement requests or configuration change notifications to the server.
- CCR_TALK: Used by the server to send annunciation messages to annunciators.

Multiple Parallel Alarm Configurations

Exactly one alarm server needs to be running for each alarm system configuration, i.e. for each alarm tree root. Clients are bound to a specific alarm configuration via a preference setting that selects their alarm tree root name. Optionally, it is possible to enable users to change the alarm configuration at runtime via a selector in the Alarm Tree display.

“Global” Alarms

The following is under development, not in operational use:

An alarm server can send information about alarms that stay un-acknowledged for a configurable time to a “GLOBAL_SERVER” alarm topic in JMS. A corresponding global alarm table user interface can display such alarms and allow operators to quickly switch to that alarm system configuration, where they then have access to the alarms guidance etc.

The use case for this is a central control room combined with several auxiliary, sub-system-specific control rooms. For example a cryogenic or conventional facility control room in addition to a bigger “main” control room. Most of the time, the auxiliary control rooms use their own alarm system configuration, specific to their needs, and all alarms are handled locally.

At night or on weekends, however, the auxiliary control rooms are not manned. At those times, alarms are not locally acknowledged, and after some timeout they will appear in the global alarm table of the main control room.

A listener to JMS messages on the GLOBAL_SERVER topic can also be used to trigger automated email notifications or send cell phone text messages in response to unhandled, i.e. un-acknowledged global alarms.

13.5. Relational Database Setup

Before using the alarm system tools, you need to create the required table structure in your RDB, see also Chapter 9, *Relational Database (RDB)*. Refer to the sources for the plugin `org.csstudio.alarm.beast`. Its `dbd` subdirectory contains schema definitions for MySQL, Oracle and PostgreSQL. Pick the file that is appropriate for your RDB, and execute the commands listed in the file.

When done, you should have the alarm tables defined in your RDB. You have to use basic RDB administration skills to create a user and password for an account that the alarm tools can use to read and modify the alarm system configuration.

13.6. Building the Tools

You need these command-line tools:

```
org.csstudio.alarm.beast.configtool/AlarmConfigTool.product
org.csstudio.alarm.beast.server/AlarmServer.product
```

For first tests, you can run both tools from within the Eclipse IDE as described in Chapter 4, *Compiling, Running, Debugging CSS*, Section 4.3, “IDE Example”, but note that you will have to provide command-line arguments to them. After first tests are successful, you can export them from the IDE as described in the same section.

Finally, you will need to include the alarm system client GUI (Alarm Table, Alarm Tree, Annunciator) into your end-user CSS product.

13.7. Authentication and Authorization

The Alarm Config Tool and the Alarm Server obviously need an account with write access to the alarm tables in the RDB because they modify the alarm configuration or state.

The alarm client GUI (Alarm Table, ...) is primiarily read-only, so a corresponding read-only RDB account would be sufficient. The GUI does, however, use CSS CSS authentication and authorization to allow a user to “Log On”, and qualified users are then permitted to modify the alarm configuration. It is in fact a prime feature of the alarm system that selected users can modify the alarm configuration on-line. For this to function, the CSS alarm GUI acutally requires an RDB account with write access. The *CSS alarm GUI should therefore be provided with a write-access RDB account*, and CSS authentication and authorization is then used to limit the use of that account within CSS.

The alarm GUI uses these Authorization IDs:

- `alarm_ack` - Users with this authorization can acknowledge alarms.
- `alarm_config` - Users with this authorization can edit the alarm system configuration.

For initial tests, you can configure CSS authentication and authorization to use a “dummy” mode where every user can authenticate and gain these rights. Eventually, however, you might want to configure proper authentication and authorization. Refer to Chapter 12, *Authentication and Authorization* for details.

13.8. Alarm System Preferences

Most of the alarm system related preference settings are explained and listed with their defaults in the file `preferences.ini` of the plugin `org.csstudio.alarm.beast`. You will have to adjust the following, most important settings settings for your site:

- URL, user, password and schema for connecting to the RDB that holds your alarm configuration. See Section 9.4, “RDB URLs and Schemata”.
- URL of your JMS server.
- Name of your alarm tree “Root”.

This is typically done by adding the site-specific settings to the `plugin_customization.ini` file of your product, see Chapter 6, *Hierarchical Preferences*. For command-line tools like the alarm config tool you can also create a file `settings.ini` as mentioned below.

13.9. Creating New Alarm Configuration, Bulk Modifications

Each alarm setup starts with a new, empty configuration in the RDB. The CSS alarm user interface will then allow to edit that configuration, but the initial, empty configuaration needs to be created in the RDB.

One way to create a new configuration is an SQL shell. Assume you want to create a configuration called “demo”, first check that it does not already exist:

```
SELECT * FROM ALARM_TREE
  WHERE NAME='demo' AND PARENT_CMPNT_ID IS NULL;
```

Find the next available component ID:

```
SELECT MAX(COMPONENT_ID) FROM ALARM_TREE;
```

Then create a new alarm tree root like this, remembering to use the **next** available component ID:

```
INSERT INTO ALARM_TREE (NAME, COMPONENT_ID)
VALUES ('demo', 1021);
```

While such direct SQL manipulations will work, it may be easier to use the `AlarmConfigTool`. The Alarm Config Tool can import alarm configurations from an XML file format. A minimal, empty configuration with name “demo” would look like this, usually saved to a file called `demo.xml`:

```
<config name="demo">
</config>
```

The sources for the plugin `org.csstudio.alarm.beast.configtool` contain a schema file for the alarm configuration XML file format, `AlarmConfigurationSchema.xsd`. This can be used as a reference or to check an alarm configuration XML file with XML tool before importing it via the alarm config tool.

To import such an XML configuration file into the RDB, issue the following command, assuming that you have a file `settings.ini` that holds your site-specific RDB connection parameters:

```
AlarmConfigTool -pluginCustomization /path/to/my/settings.ini \
-import -root demo -file demo.xml
```

The config tool has more command-line options to list available configurations or to export a configuration to a file:

```
# Display all parameters
AlarmConfigTool -help

# List alarm tree root names
AlarmConfigTool -list

# Export alarm configuration to XML file
AlarmConfigTool -export -root demo -file demo.xml
```

While it is usually most convenient to edit an alarm configuration from the alarm client user interface, bigger changes to the alarm configuration are sometimes easier by exporting the existing configuration to an XML file, editing it, then re-importing it. The AlarmConfigTool is meant to handle such bulk-imports of the complete alarm configuration.

To perform partial changes of the alarm configuration, the AlarmConfigTool has the `-delete` option to remove a subtree of the configuration. You need to specify the complete path to the element that ought to be removed:

```
AlarmConfigTool -root demo -delete /demo/area1/system2
```

If is possible to modify an existing configuration via the `-modify` command. It will load an XML file and add newly listed items from the XML file to the alarm configuration, or update the guidance messages, related displays etc. of existing items in the configuration. It can not move items from existing locations to new locations in the tree, and you cannot add the same trigger PV name to different sections of the tree. For such wider ranging changes you will have to export, edit and then re-import the complete configuration.

Note that the AlarmConfigTool only updates the database! If you are already using the alarm configuration, you should stop the AlarmServer and CSS alarm displays before performing such database updates, then start them back up when the database modifications are completed. If you update the alarm configuration from the CSS alarm GUI, all parts of the alarm system are notified about these changes so that they can update accordingly. Direct changes to the alarm database via SQL access or the AlarmConfigTool, however, go unnoticed by the AlarmServer and GUI clients, so you need to restart them.

13.10. Putting it all together

So far, we described the alarm system in general as well as the setup of its infrastructure. In the following sections, we will describe how to run the alarm server and how to use the alarm client GUI.

Since this is a distributed system, it is important to remember its components:

- Control System: Often overlooked when you create your first alarm system test setup, you need for example an EPICS soft IOC that creates alarms.
- Configuration in RDB: Previous sections described the RDB table setup and how to use the Alarm Config Tool to create an initial alarm system configuration.
- JMS Server: See Chapter 11, *Java Message Server* how to start JMS. The Alarm Server and the Alarm GUI need to be configured with the URL of your JMS server to communicate.
- Alarm Server: Its operation will be described in the following sections.
- CSS Alarm GUI: Your CSS product needs to include the alarm tree and alarm table plugins to allow access to the alarm system. In addition, at least “dummy” authentication and authorization needs to be configured to permit changes to the alarm configuration from the client GUI. See Chapter 12, *Authentication and Authorization*.

13.11. Alarm Server

The Alarm Server reads an alarm configuration, monitors the PVs of that configuration and notifies alarm clients about changes in the alarm state. It persists the alarm state in the RDB. For example, information about latched alarms is written to the RDB. When the Alarm Server is stopped and re-started, it will initialize from the RDB and learn about the previous state of all alarms. This way a previously latched alarm is recognized and not reported as a new alarm.

The Alarm Server is a command-line tool that is configured via Eclipse preferences, i.e. usually via a customization file like this:

```
AlarmServer -consoleLog -pluginCustomization /path/to/alarm_server.ini
```

The alarm server plugin `org.csstudio.alarm.beast.server` includes an example plugin customization file `plugin_customization.ini`:

```
# Alarm System 'root', i.e. configuration name
```

```
org.csstudio.alarm.beast/root_component=Annunciator

# Alarm System RDB Connection
org.csstudio.alarm.beast/rdb_url=jdbc:mysql://localhost/alarm
org.csstudio.alarm.beast/rdb_user=alarm
org.csstudio.alarm.beast/rdb_password=$alarm
org.csstudio.alarm.beast/rdb_schema=ALARM

# Alarm System JMS Connection
org.csstudio.alarm.beast/jms_url=failover:(tcp://localhost:61616)
org.csstudio.alarm.beast/jms_user=alarm
org.csstudio.alarm.beast/jms_password=$alarm

# Alarm Server: Period for repeated annunciation of active alarms
org.csstudio.alarm.beast.server/nag_period=00:15:00

# Channel Access
# Network traffic can be optimized by only monitoring ALARM updates
org.csstudio.platform.libs.epics/use_pure_java=false
org.csstudio.platform.libs.epics/monitor=ALARM
org.csstudio.platform.libs.epics/addr_list=127.0.0.1

# Logging preferences
org.csstudio.logging/console_level=CONFIG
org.csstudio.logging/jms_url=
```

For more on the Eclipse preference mechanism, see Chapter 6, *Hierarchical Preferences*.

If you started the Alarm Server successfully, its terminal output should resemble this, displaying the JMS topics used by the alarm server:

```
Alarm Server 3.0.0
Configuration Root: demo
JMS Server Topic:   demo_SERVER
JMS Client Topic:   demo_CLIENT
JMS Talk Topic:     demo_TALK
JMS Global Topic:   GLOBAL_SERVER
Read 50003 PVs in 1.69 seconds: 29589.0 PVs/sec
```

The alarm server communicates with the alarm GUI via JMS messages described in the following section. The alarm GUI will either show alarm updates, or it will indicate a server timeout if the alarm server does not communicate. To debug the setup, the JMS Monitor (Chapter 31, *JMS Monitor - org.csstudio.debugging.jmsmonitor*) can be used to monitor the JMS server, client and talk topics. The JMS server topic should show alarm state changes or at least periodic IDLE messages.

13.12. Alarm System JMS Message Types

The Alarm Server sends this type of JMS `MapMessage` for alarm state changes:

- `TYPE`: Set to “alarm” to identify as alarm message.
- `TEXT`: STATE in normal mode, STATE_MAINTENANCE in maintenance mode.
- `CONFIG`: Name of the alarm configuration, i.e. alarm root.
- `NAME`: PV name that has changed alarm state
- `CURRENT_SEVERITY`: Current severity of the PV.

- `CURRENT_STATUS`: Current status of the PV.
- `SEVERITY`: Alarm severity of the PV. For latched or acknowledged alarms, this can differ from the current severity.
- `STATUS`: Alarm status of the PV.
- `EVENTTIME`: Time of the original alarm, i.e. when SEVERITY became active.
- `APPLICATION_ID`: AlarmServer.
- `HOST`: Host name running the alarm server.
- `USER`: Name of user who is running the alarm server.

In the absense of alarm state changes, the Alarm Server sends an IDLE message, by default every 10 seconds. If the alarm client GUI does not see any message from the alarm server for twice the IDLE period, it declares a server timeout.

- `TYPE`: Set to “alarm” to identify as alarm message.
- `TEXT`: IDLE in normal mode, IDLE_MAINTENANCE in maintenance mode.
- `CONFIG`: Name of the alarm configuration, i.e. alarm root.
- `APPLICATION_ID`: AlarmServer.
- `HOST`: Host name running the alarm server.
- `USER`: Name of user who is running the alarm server.

Alarm clients send this message to acknowledge or un-acknowledge an alarm:

- `TYPE`: Set to “alarm” to identify as alarm message.
- `TEXT`: ACK respectively UN-ACK.
- `CONFIG`: Name of the alarm configuration, i.e. alarm root.
- `NAME`: PV name
- `APPLICATION_ID`: CSS.
- `HOST`: Host name running CSS.
- `USER`: Name of user who is running CSS.

Alarm clients send this message to enable or disable maintenance mode:

- `TYPE`: Set to “alarm” to identify as alarm message.
- `TEXT`: MODE.
- `VALUE`: MAINTENANCE or NORMAL.
- `CONFIG`: Name of the alarm configuration, i.e. alarm root.
- `APPLICATION_ID`: CSS.
- `HOST`: Host name running CSS.
- `USER`: Name of user who is running CSS.

After modifying the alarm configuration, alarm clients send this message to the server:

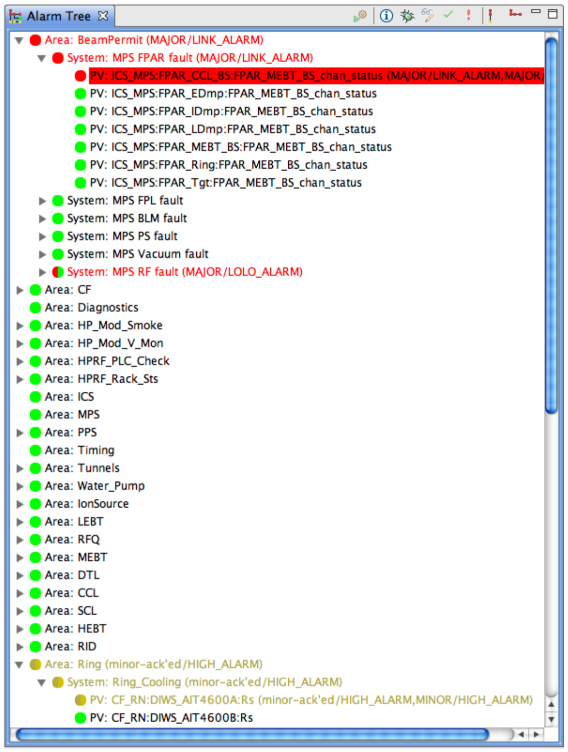
- `TYPE`: Set to “alarm” to identify as alarm message.
- `TEXT`: CONFIG.
- `CONFIG`: Name of the alarm configuration, i.e. alarm root.
- `NAME`: Path to the modified configuration item.
- `APPLICATION_ID`: CSS.
- `HOST`: Host name running CSS.
- `USER`: Name of user who is running CSS.

Note that other clients are also listening to this message, so the server as well as other clients will react by updating their configuration from the RDB. If the path contains a specific configuration item like "/area/system/subsystem/pv", only the configuration for that item needs to be updated from the RDB. If the path is empty, a bigger change to the configuration means that the complete alarm configuration needs to be updated from the RDB.

13.13. Alarm Tree View

The Alarm Tree display can be opened from the menu `CSS`, `Alarm`, `Alarm Tree`. It is the primary display for viewing and configuring the complete alarm tree.

Figure 13.2. Alarm Tree



```
Alarm Tree
▼ Area: BeamPermit (MAJOR/LINK_ALARM)
    ▼ System: MPS FPAR fault (MAJOR/LINK_ALARM)
        PV: ICS_MPS:FPAR_CCL_BS:FPAR_MEBT_BS_chan_status (MAJOR/LINK_ALARM,MAJOR/
        PV: ICS_MPS:FPAR_EDmp:FPAR_MEBT_BS_chan_status
        PV: ICS_MPS:FPAR_IDmp:FPAR_MEBT_BS_chan_status
        PV: ICS_MPS:FPAR_LDmp:FPAR_MEBT_BS_chan_status
        PV: ICS_MPS:FPAR_MEBT_BS:FPAR_MEBT_BS_chan_status
        PV: ICS_MPS:FPAR_Ring:FPAR_MEBT_BS_chan_status
        PV: ICS_MPS:FPAR_Tgt:FPAR_MEBT_BS_chan_status
    ▶ System: MPS FPL fault
    ▶ System: MPS BLM fault
    ▶ System: MPS PS fault
    ▶ System: MPS Vacuum fault
    ▶ System: MPS RF fault (MAJOR/LOLO_ALARM)
▶ Area: CF
  Area: Diagnostics
▶ Area: HP_Mod_Smoke
▶ Area: HP_Mod_V_Mon
▶ Area: HPRF_PLC_Check
▶ Area: HPRF_Rack_Sts
  Area: ICS
  Area: MPS
▶ Area: PPS
  Area: Timing
▶ Area: Tunnels
▶ Area: Water_Pump
▶ Area: IonSource
▶ Area: LEBT
▶ Area: RFQ
▶ Area: MEBT
▶ Area: DTL
▶ Area: CCL
▶ Area: SCL
▶ Area: HEBT
▶ Area: RID
▼ Area: Ring (minor-ack'ed/HIGH_ALARM)
    ▼ System: Ring_Cooling (minor-ack'ed/HIGH_ALARM)
        PV: CF_RN:DIWS_AIT4600A:Rs (minor-ack'ed/HIGH_ALARM,MINOR/HIGH_ALARM)
        PV: CF_RN:DIWS_AIT4600B:Rs
```

The alarm tree displays the hierarchical structure of the alarm system. Alarms are arranged into

- Areas - Top-level elements of the alarm hierarchy.
- Systems, Subsystems - Below the top-level areas, there can be multiple levels of systems and subsystems to organize your alarm configuration.
- PVs - Finally, the alarm trigger PVs generate actual alarms.

It is suggested to use physical areas, because that way the general localization of an alarm in the machine is obvious. The systems or subsystems could for example be Vacuum or Cooling, i.e. again physical systems of your machine.

When configuring the guidance or related displays of an item in the alarm configuration, this will affect all entries below the respective item. For example, when adding a display link to an area, this link will be available for each system, subsystem and PV below that area. This way it is easy to assign overview displays for an area, or add guidance with contact information for a system to those elements in the alarm tree without need to duplicate the information for each PV.

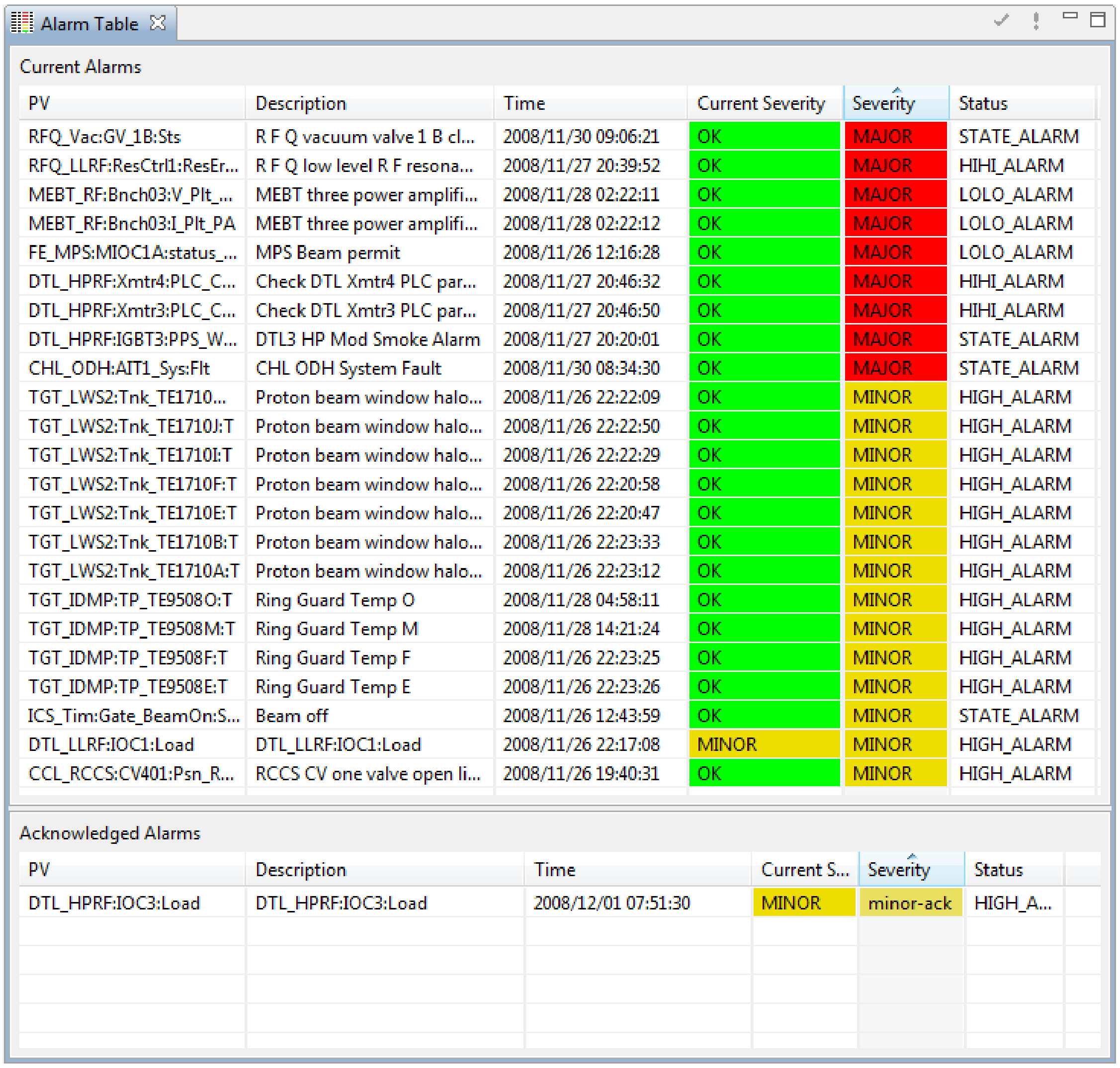
The alarm tree needs to be used to add, rename or remove entries of the alarm configuration. To operators, the alarm tree can be useful if many alarms are active, because its hierarchical view can allow operators to identify the affected areas or subsystems better than a plain list of alarms could do.

For details on how to use the alarm tree, refer to its online help.

13.14. Alarm Table View

In an ideal operational setup, there are only very few alarms. If alarms trigger, they may be from very different areas of the machine. In this scenario, operators are often not interested in all the possible alarms as they are displayed by the Alarm Tree. Instead, they only need to know the currently active alarms.

Figure 13.3. Alarm Table



Current Alarms

PV	Description	Time	Current Severity	Severity	Status
RFQ_Vac:GV_1B:Sts	R F Q vacuum valve 1 B cl...	2008/11/30 09:06:21	OK	MAJOR	STATE_ALARM
RFQ_LLRF:ResCtrl1:ResEr...	R F Q low level R F resona...	2008/11/27 20:39:52	OK	MAJOR	HIHI_ALARM
MEBT_RF:Bnch03:V_Plt_...	MEBT three power amplifi...	2008/11/28 02:22:11	OK	MAJOR	LOLO_ALARM
MEBT_RF:Bnch03:I_Plt_PA	MEBT three power amplifi...	2008/11/28 02:22:12	OK	MAJOR	LOLO_ALARM
FE_MPS:MIOC1A:status_...	MPS Beam permit	2008/11/26 12:16:28	OK	MAJOR	LOLO_ALARM
DTL_HPRF:Xmtr4:PLC_C...	Check DTL Xmtr4 PLC par...	2008/11/27 20:46:32	OK	MAJOR	HIHI_ALARM
DTL_HPRF:Xmtr3:PLC_C...	Check DTL Xmtr3 PLC par...	2008/11/27 20:46:50	OK	MAJOR	HIHI_ALARM
DTL_HPRF:IGBT3:PPS_W...	DTL3 HP Mod Smoke Alarm	2008/11/27 20:20:01	OK	MAJOR	STATE_ALARM
CHL_ODH:AIT1_Sys:Flt	CHL ODH System Fault	2008/11/30 08:34:30	OK	MAJOR	STATE_ALARM
TGT_LWS2:Tnk_TE1710...	Proton beam window halo...	2008/11/26 22:22:09	OK	MINOR	HIGH_ALARM
TGT_LWS2:Tnk_TE1710J:T	Proton beam window halo...	2008/11/26 22:22:50	OK	MINOR	HIGH_ALARM
TGT_LWS2:Tnk_TE1710I:T	Proton beam window halo...	2008/11/26 22:22:29	OK	MINOR	HIGH_ALARM
TGT_LWS2:Tnk_TE1710F:T	Proton beam window halo...	2008/11/26 22:20:58	OK	MINOR	HIGH_ALARM
TGT_LWS2:Tnk_TE1710E:T	Proton beam window halo...	2008/11/26 22:20:47	OK	MINOR	HIGH_ALARM
TGT_LWS2:Tnk_TE1710B:T	Proton beam window halo...	2008/11/26 22:23:33	OK	MINOR	HIGH_ALARM
TGT_LWS2:Tnk_TE1710A:T	Proton beam window halo...	2008/11/26 22:23:12	OK	MINOR	HIGH_ALARM
TGT_IDMP:TP_TE9508O:T	Ring Guard Temp O	2008/11/28 04:58:11	OK	MINOR	HIGH_ALARM
TGT_IDMP:TP_TE9508M:T	Ring Guard Temp M	2008/11/28 14:21:24	OK	MINOR	HIGH_ALARM
TGT_IDMP:TP_TE9508F:T	Ring Guard Temp F	2008/11/26 22:23:25	OK	MINOR	HIGH_ALARM
TGT_IDMP:TP_TE9508E:T	Ring Guard Temp E	2008/11/26 22:23:26	OK	MINOR	HIGH_ALARM
ICS_Tim:Gate_BeamOn:S...	Beam off	2008/11/26 12:43:59	OK	MINOR	STATE_ALARM
DTL_LLRF:IOC1:Load	DTL_LLRF:IOC1:Load	2008/11/26 22:17:08	MINOR	MINOR	HIGH_ALARM
CCL_RCCS:CV401:Psn_R...	RCCS CV one valve open li...	2008/11/26 19:40:31	OK	MINOR	HIGH_ALARM

Acknowledged Alarms

PV	Description	Time	Current S...	Severity	Status
DTL_HPRF:IOC3:Load	DTL_HPRF:IOC3:Load	2008/12/01 07:51:30	MINOR	minor-ack	HIGH_A...

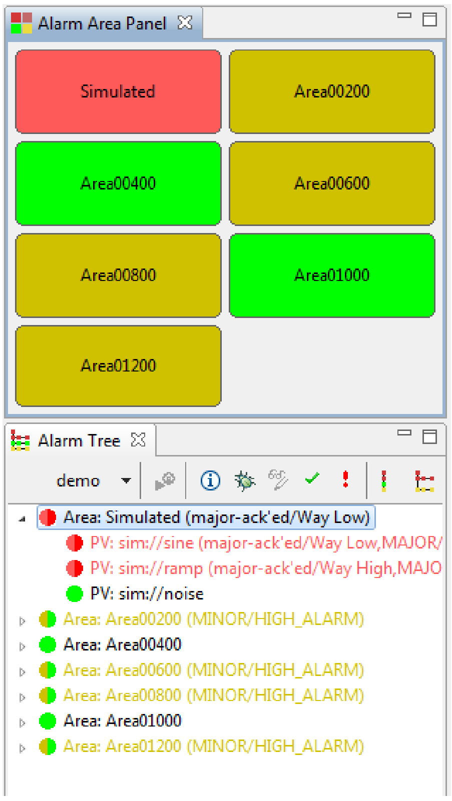
The Alarm Table display is the primary operator interface to the alarm system in an operational setup for a machine that is generally running fine. Most of the time, the alarm table will be empty, because there are no alarms. If alarms occur, the alarm table allows operators to inspect them quickly.

For details on how to use the alarm table, refer to its online help.

13.15. Alarm Area Panel

The Alarm Area Panel can be used as a top-level display of the current alarm state. A preference setting of the panel is used to configure which level of the alarm tree hierarchy it should display.

Figure 13.4. Alarm Area Panel



```
Area: Simulated (major-ack'ed/Way Low)
    PV: sim://sine (major-ack'ed/Way Low,MAJOR/
    PV: sim://ramp (major-ack'ed/Way High,MAJO
    PV: sim://noise
Area: Area00200 (MINOR/HIGH_ALARM)
Area: Area00400
Area: Area00600 (MINOR/HIGH_ALARM)
Area: Area00800 (MINOR/HIGH_ALARM)
Area: Area01000
Area: Area01200 (MINOR/HIGH_ALARM)
```

It will typically be used to display the first level of the alarm hierarchy, that is all “Area” components of the alarm tree. For each area, it displays the name, coloring the panel to indicate the alarm state of that area as in Figure 13.4, “Alarm Area Panel”.

The tool can also be used to display a single panel with the alarm state of the overall alarm tree, i.e. the root element, by configuring it to use level 0 of the alarm tree hierarchy. When set to level 2 of the alarm tree, it will display all “System” components. With level 3 it would display all “Sub-System” components, but this and higher alarm tree components are not useful in practice.

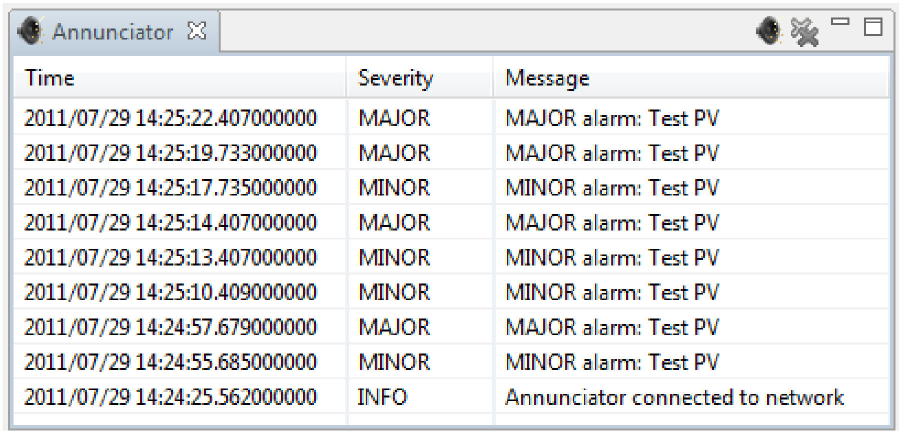
For details on how to use the alarm table, refer to its online help.

13.16. Annunciator

This tool performs voice annunciation of alarms. It usually listens to the JMS `..._TALK` topic associated with an alarm tree configuration and speaks received alarms to the operator.

The annunciator can be configured to listen to one or more topics. It will typically listen to one specific alarm tree `..._TALK` topic like `CCR_TALK`, but in principle it can listen to talk topics from multiple alarm tree configurations. It could also annunciate messages that are sent to JMS topics by means other than the alarm system.

Figure 13.5. Annunciator View



Time	Severity	Message
2011/07/29 14:25:22.407000000	MAJOR	MAJOR alarm: Test PV
2011/07/29 14:25:19.733000000	MAJOR	MAJOR alarm: Test PV
2011/07/29 14:25:17.735000000	MINOR	MINOR alarm: Test PV
2011/07/29 14:25:14.407000000	MAJOR	MAJOR alarm: Test PV
2011/07/29 14:25:13.407000000	MINOR	MINOR alarm: Test PV
2011/07/29 14:25:10.409000000	MINOR	MINOR alarm: Test PV
2011/07/29 14:24:57.679000000	MAJOR	MAJOR alarm: Test PV
2011/07/29 14:24:55.685000000	MINOR	MINOR alarm: Test PV
2011/07/29 14:24:25.562000000	INFO	Annunciator connected to network

The annunciator is available in two forms:

- Eclipse View: The Annunciator view can be opened from the CSS Alarm menu as long as your CSS product includes the plugin `org.csstudio.alarm.beast.annunciator`.
- Headless Application: The Annunciator application is started from the command line. To create the headless executable, export the `Annunciator.product` product from the annnuciator plugin. The headless annunciator application is also referred to as JMS2SPEECH.

The annunciator view is convenient for use within CSS and has the added advantage of displaying a list of recently received alarm messages. It will, however, *stop annunciating* as soon as the annnuciator view is closed or hidden. It is therefore important to keep the annunciator view visible to assert that it can perform annunciations.

The annunciator view offers two buttons it its toolbar: One to temporarily silence alarms, the other to clear the list of past annunciations.

The headless Annunciator application needs to be launched separately from the CSS user interface, but has the advantage that it will annunciate even when the CSS user interface is closed. For end users, the annunciator view may be more convenient, but in a control room it is suggested to run at least one copy of the headless Annunciator application.

The annunciation of a message takes some time. When several messages are received for annunciation, they are ordered by their severity. When too many messages arrive, especially from the alarm system, there is usually little use in annunciating them because the bigger message is: There are MANY messages, do something about it! If the queue length exceeds a configurable threshold, the queue is therefore cleared and only a single “There are ... more messages” statement is announced.

Preferences

Either version of the annunciator is configured via preferences. Refer to the `preferences.ini` file of the annunciator plugin or the CSS preference GUI when using the view to configure the following, where the first two are essential for the operation of the tool:

- URL of the JMS server from which the annunciator receives messages.
- JMS topics to which it listens. This is a comma-separated list like `CCR_TALK, Demo_TALK`
- Translations file. This file can be used to provide pronunciation help. Refer to the online help for details.
- List of message severties, separated by comma, ordered by priority (highest first) to define how the annunciator should prioritize when receiving several concurrent messages.

- Threshold for ignoring flurry of messages. When more than this number of messages queue up, a shorter “There are .. more messages” statement is annunciated.
- The number of messages kept in the Annunciator View display of recent messages.

Also note the separate preferences of the speech library mentioned below.

Message Texts

Fundamentally, messages are annunciated as received. The alarm server will usually include the alarm severity in the message. For example, an alarm with description

```
Low Water Pressure
```

will be annunciated as

```
"MINOR alarm: Low Water Pressure"
```

when it enters a MINOR alarm state.

If an alarm description starts with an asterisk as in

```
*Low Water Pressure
```

the alarm server will use the description as given. It will *not* prefix it with the alarm severity. A PV with above Description thus results in just announcing

```
"Low Water Pressure"
```

In addition, descriptions starting with an asterisk can include the following formatting elements:

- The text `{0}` is replaced with the current alarm severity.
- The text `{1}` is replaced with the value of the trigger PV that caused the alarm.

For example, a PV with description

```
*{0} alarm: Low Water Pressure
```

would again result in an annunciation

```
"MINOR alarm: Low Water Pressure"
```

The effect would be the same as having used the default annunciation format without `*`.

By using the `*` format, however, you can also use descriptions like

```
*{0} water alarm, level is {1} gallons
```

which will result in an annunciation similar to

```
"MINOR water alarm, level is 3.142 gallons"
```

The format elements can be in any order. For example, the value can be used before the alarm severity. The description

```
*Water below {1} gallons, {0} alarm
```

will result in an annunciation similar to

```
"Water below 3.142 gallons, MAJOR alarm"
```

We mentioned that messages will be suppressed when too many messages arrive. Very important messages can be prefixed with an initial exclamation mark:

```
!Running low on cookies
```

Such messages will always be annunciated and not be dropped when there too many other messages.

When combining the formatting element `*` and the `!` to prevent suppression, note the required order. You have to use

```
*!My description ...
```

i.e. first the `*`, then the `!`.

Speech Systems

The annunciator uses the plugin `org.csstudio.utility.speech` for the text-to-speech conversions. That plugin can be configured to use either the pure Java FreeTTS library that it provides, an external tool, or a UDP server.

Refer to the `preferences.ini` of the utility.speech plugin for available options.

When selecting the external tool, a program called `say` is expected to be found on the PATH. It must accept the text to annunciate as a command line argument. On Mac OS X, such a `say` command is already part of the operating system. On Linux computers, one can often use the following shell script to invoke festival:

```
#!/bin/sh
#
# Example 'say' script for Linux

echo "$@" | festival --tts
```

When selecting the UDP server, you also need to configure the hostname or boradcast address and port number of the UDP server, again refer to the `preferences.ini` of the utility.speech plugin. The UDP server will receive each annunciation as a seprate UDP packet.

Active Alarms Annunciation

The alarm server will issue one annunciation for each new alarm that is configured to be annunciated. If operators miss an annunciation, we generally assume that they were distracted by more important tasks. Once they have time, they can inspect the alarm GUI for new alarms, for example from the time-sorted alarm table, or consult the list of recent messages in the annunciator.

Annunciations are not repeated, because in general we try to minimize the noise generated by the alarm system. Sometimes, however, it can be useful to remind operators that there were alarms that they might have missed.

The `nag_period` configuration parameter of the alarm server instructs the alarm server to generate periodic voice annunciations

```
"There are ... active alarms"
```

whenever there are active, that is un-acknowledged alarms. The default `nag_period` of `00:15:00` causes this annunciation to be performed every 15 minutes.

Whenever operators acknowledge or un-acknowledge an alarm, or change the alarm configuration, or if there is a different annunciation, the timer for active alarm reminders is reset. In normal operation of the alarm system where operators acknowledge alarms, or while there are other annunciations that we assume are heard by operators, there will therefore be no reminder. On the other hand, if operators have not interacted with the alarm system, not acknowledged any alarms, now were there any other new annunciations, yet there are still active alarms, this reminder will help operators every 15 minutes to notice that something happened that they might have missed.

The active alarm reminders can be disabled by configuring the `nag_period` as `0`.

13.17. JMS Alarm Log

The JMS2RDB tool described in Chapter 33, *RDB Logging - org.cstudio.sns.jms2rdb* can be used to log all alarm traffic to the RDB. This way, detail of alarm state changes and operator acknowledge actions will be available in the CSS log for later analysis. The message history display described in Chapter 29, *Message History Browser - org.cstudio.alarm.beast.msghist* allows direct access to the log, although it can be time consuming the decode the raw messages.

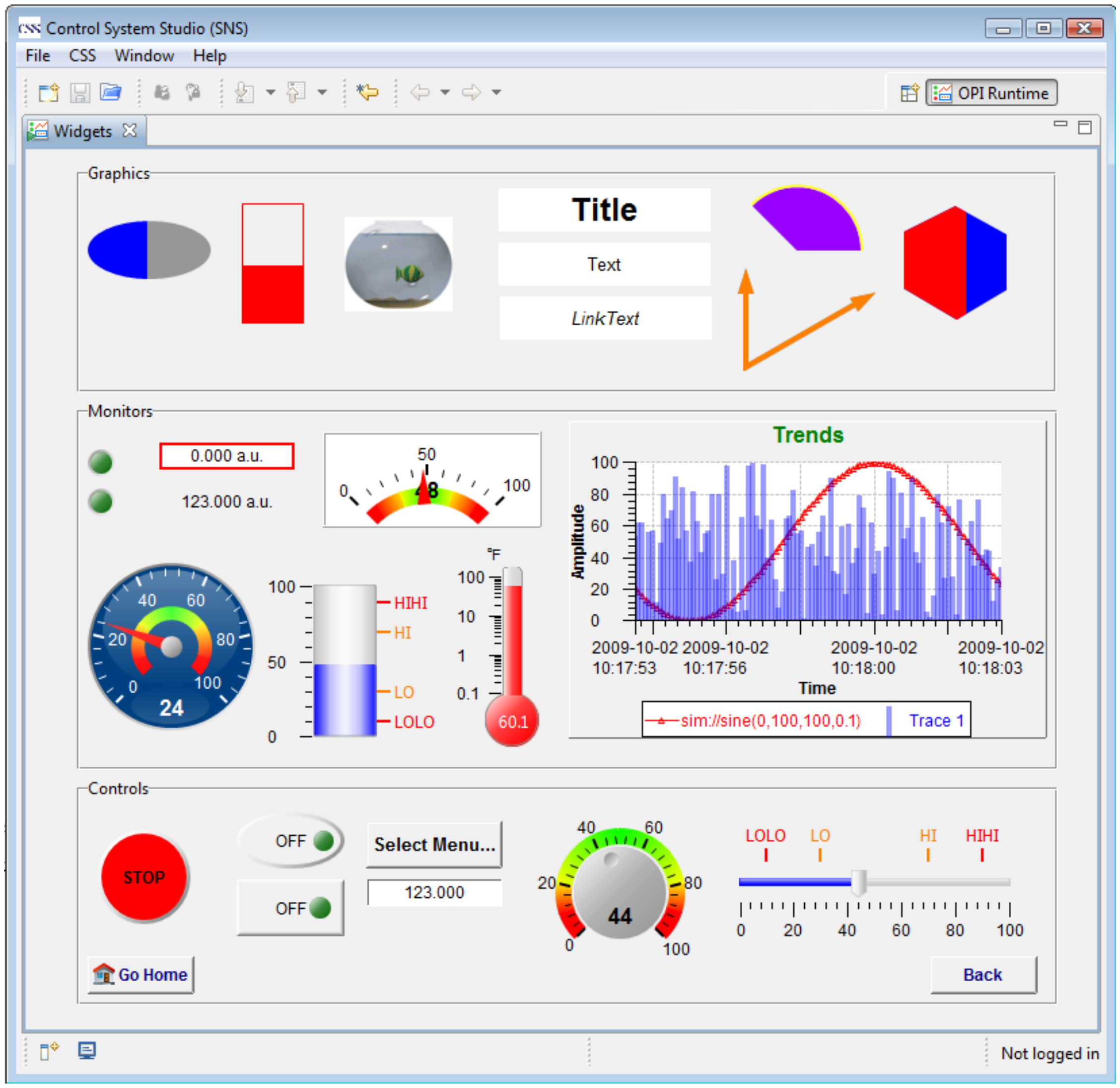
At the SNS, a JSP-based collection of reports can display how many alarms were active over the last days or weeks, which alarms represent the “Top 10” of the most frequent alarms, and how one alarm behaved in detail, including histograms of alarm durations.

This reporting package, however, is currently part of a bigger, more SNS-specific reporting package. Contact Kay Kasemir if you are interested in collaborating on a more portable version of these reports.

Chapter 14. BOY Operator Display

The Best OPI, Yet (BOY) is an operator interface editor and runtime. To end users, this might be the most interesting component of CSS. It can be the primary user interface to the control system: Start CSS, then open BOY displays to monitor and control the system. BOY also makes it fairly easy for end-users to create their own operator interface displays.

Figure 14.1. Example of BOY Widgets



BOY includes extensive online help that covers everything from first steps to extending BOY by implementing your own custom widgets. Go through the installation of the example display files and the “Getting Started” section of the online help, because this chapter will not duplicate that information. Instead, we concentrate on setup suggestions.

14.1. Setup

To use BOY, your CSS product needs to include the `org.csstudio.opibuilder.*` plugins and their dependencies. The converter plugins described in Section 14.3, “Converters” are optional.

You should prepare the following files for your site and place them in a location that is accessible by all users.

- Color definitions - A file that defines color macros.
- Font definitions - A file that defines font macros.
- Schema - An `*.opi` file that defines defaults for widget properties.
- Top OPIs - A list of `*.opi` shortcuts.

The definition of color and font macros allows you to create uniform display files. By using a macro `Background` for the display background color and a macro `Title` for the font of a title label your display files will appear consistent. In addition, you can later adjust colors and fonts by simply updating the definition file instead of having to edit each individual display.

You can start with the files that are included in the BOY examples, i.e. `/BOY Examples/color.def` and `/BOY Examples/font.def`, but copying them into a location is accessible by all CSS installations at your site.

The files could be in a network file system location, but since the detailed path name syntax for such shared file system location often differs between operating systems, a web location might be more practical. BOY can read files from `http://`, `https://` and `ftp://` URLs.

Note that the fonts listed in the font definition file need to be fonts that are actually available on the computer that executes CSS, but available fonts differ between operating systems. The font definition file allows you to specify different values depending on the operating system, for example

```
// Title Font for Linux GTK
Title(linux_gtk) = Sans-bold-18
// Title Font for MacOS
Title(macosx) = Lucida Grande-bold-18
```

A practical solution for display files that need to look the same across platforms can be the use of Microsoft Office fonts. Windows and Mac OS X computers that have Microsoft Office installed already provide these fonts, and they are also available for Linux, see http://corefonts.sourceforge.net. But even with same fonts installed there can be slight differences. For example, the OS X version of Microsoft Office fonts seem to use a different size, for which you can compensate in the font definition file by using OS-specific settings:

```
// Title Font for Windows and Linux
Default=Verdana-regular-10
Default(linux_gtk)=Verdana-regular-10
// OS X needs different size to get same look
Default(macosx_cocoa)=Verdana-regular-14
```

The Scheme file allows you to define the defaults for widget properties in newly created OPI files. For example, you can use a schema file to use macro names for the default fonts and colors of widgets.

The default for the Top OPIs will point to the BOY example files like `/BOY Examples/main.opi`. You probably want to adjust them to load certain top-level OPI files of your site, like `http://my_opi_server/opis/start_screen.opi`

14.2. Team Support

By including Eclipse “Team” support like the Concurrent Versions System (CVS) or support for another version system, you can keep your OPI files in a software repository, then check them out into the CSS workspace, edit them there, and commit changes back to the repository.

14.3. Converters

The plugin `org.csstudio.opibuilder.converter` can convert `*edl` displays of the EPICS Extensible Display Manager (EDM) to the BOY `*.opi` file format, and `org.csstudio.opibuilder.adl2boy` performs this for `*.adl` displays of MEDM.

Such conversions are naturally limited because a converter can only attempt a basic translation of for example rectangles in an EDM display into rectangles in a BOY display. It cannot tell if the rectangles in the EDM display were meant to create a visual group of widgets that are better transformed into a Grouping Container widget in BOY.

Chapter 15. Site-Specific Products

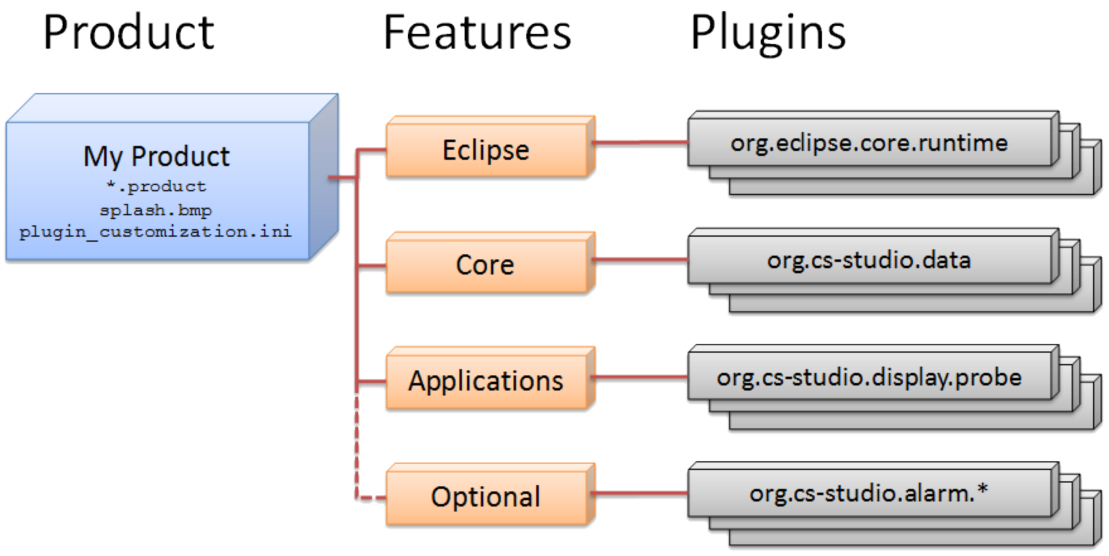
This chapter describes how and why Plug-ins get combined into a Product that is specific for a site.

As already explained in Section 2.2, “Java, Eclipse, RCP”, a product combines selected Plug-ins with configuration files and an OS-specific launcher.

There are several CSS products: Archive Engine, Archive Config Tool, Alarm Server, Alarm Config Tool are all examples for CSS products. When we talk about **The CSS Product**, however, we usually refer to the CSS user interface product that the end users see. A product that is just called `css` or maybe `css-xyz` because it has been pre-configured for users at an institute called XYZ. A product that includes Probe, Data Browser, maybe the alarm table and tree displays.

Why is there more than one CSS product? Why can it be difficult to select the required Plug-ins?

Figure 15.1. Composition of a Product



15.1. Site-Specific Plugin Selection and Settings

Plug-ins contain the Java code and associated content like online help for some CSS functionality. One example would be a `databrowser` Plug-in that implements the Data Browser. System integrators at different sites might prefer some Plug-ins over others, for example one operator interface tool over another.

A product is a collection of the plugins chosen for use at a site. A product definition file can in fact directly list the desired plugins. In practice, however, plugins are usually first combined into *Features*, and the product is then assembled from these features. Features allow grouping of plugins by functionality. This way it is easier to add or remove a certain functionality from the product.

For example, an *optional feature* can list plugins that are not needed by every user of a product. When building the product, such an optional feature may be excluded from the product, but it is made available in an online update repository. Users who down-load the product can then add the optional feature as desired via the `Help`, `Install New Software ...` menu.

Products also contain the default settings, the `plugin_customization.ini` file described in Chapter 6, *Hierarchical Preferences*. To users of CSS it is most convenient when the suitable settings for their site, for example EPICS Channel Access addresses, web links, LDAP server hosts for authentication are already “built in”, so that there is usually no need to adjust any preference settings after installing CSS onto their office computer.

When a site needs basically the same collection of plugins, i.e. one product, but with different settings to support for example different networks where the product is used, there are two options:

1. Build different products.

   These products are assembled from the same features and plugins, but since they include different `plugin_customization.ini` files, they are in the end different.

   This option might be most convenient to end users because they have a product that works for them out of the box, but is more work for the maintainer of these products.
2. Build one product with several `*.ini` files.

   Since fundamentally only different versions of the `plugin_customization.ini` are needed, simply include several in the product plugin:
   - `plugin_customization.ini` - Shared settings.
   - `main_control_room.ini` - Adjustments for the main control room.
   - `test_network.ini` - Adjustments for the test network.
   - ... maybe more.

   These configuration files can be offered as separate downloads from the web site, or they can be included in the product plugin. In the latter case, it can be helpful to keep the product plugin un-packed:

   In the feature that adds the product plugin to the overall product configuration, assert that the product plugin is not packed into a `*.jar` file but left as an unpacked directory. In the feature editor this is done by checking the option to “Unpack the plug-in archive after the installation”. In the feature file itself, the product plugin should then **not** include the option

   ```
   unpack="false"
   ```

   Either way, users now have various `*.ini` files available and can run the product with the appropriate one from the command line:

   ```
   css -pluginCustomization /path/to/e.g./test_network.ini
   ```

   You can also prepare batch files or shell scripts that start the product with these customization files.

   This option is more convenient for the maintainer of the product because it remains one product, but requires end users to run CSS with the correct `*.ini` file resp. starter script.

So it may be obvious by now that most sites need a custom-built CSS product to provide a site-specific selection of plugins with suitable default settings. There is one more reason to publish your own product:

CSS instances can self-update from a repository. Users will see a notice that updates are available, and CSS will restart after installing them. In an operational setup it is typically of advantage to control which updates become available when and how, which means that each site that uses CSS will need its own, local update server from which its CSS instances pull updates that apply to their local CSS product.

15.2. Plug-in Dependencies

A plug-in can depend on other plug-ins. For example, a data plotting plug-in depends on others which provide access to archived data, to live data, and it also depends on a plotting plugin. When adding a plotting plugin to a product, such direct dependencies are obvious and Eclipse can help to add them to the product. There are, however, additional dependencies that Eclipse cannot automatically determine.

For example, the plug-in on which the plotting tool directly depends to read archived data only defines the programming interface for reading historic data. CSS is designed to support multiple sources of archived data, for example the XML-RPC network data server of the Channel Archiver but also an RDB-based archive. Some sites use the former, others the latter, some might use both and other sites might use an entirely different archive data store. Similar examples exist for access to an electronic log-book, or to reading live control system data.

Eclipse cannot automatically decide which implementing plug-in are necessary at a site, so it is up to the creator of a CSS product to select among the available plug-ins that implement access to archived data, a log book or live data. She might even need to implement a new, site-specific way of reading historic data or writing to a log book.

15.3. Features

As just described, we often need more than one plug-in to provide a certain functionality, for example the `databrowser` plug-in with its immediate dependencies combined with a site-specific selection of log book, live and historic data implementations. Directly listing all plug-ins in a product configuration would result in a long list that is hard to maintain.

A feature is simply a list of related plug-ins, for example all plug-ins that a site uses for the Data Browser functionality. Features can also be used to modularize the head-less built, and features can appear as separate, optional components in an update repository, allowing end users to install them into their CSS product on demand.

15.4. Creating a Product

We will now walk through the steps of creating a custom product. In the end, there will only be very little code. After all the whole point of CSS is that you can use existing plugins without having to implement everything yourself.

But unfortunately there are many pitfalls when assembling a product. It can be helpful to start small, for example try to assemble a product that only includes Probe, and get that to function as desired. Adding many more plugins for the Data Browser, operator interface etc. will then be comparably easy.

Fundamentally, an RCP Product is a plugin that implements the `org.eclipse.core.runtime.applications` extension point, and has a `*.product` file. The application extension point represents the “main” routine of the program, and the product file lists all plugins that you want to include in your product.

A CSS application should create a workbench window with certain menus into which other CSS plugins can then add their entries. Similarly, it needs to create a skeleton for the online help and preference system. Existing CSS plugins can be used to provide these.

New Product Plug-in

Create a new plugin. The suggested name is `org.csstudio.` followed by your site name and ending in `.product`, for example `org.csstudio.mysite.product`.

Add these dependencies:

`org.eclipse.core.runtime`	Defines the application extension point.

org.eclipse.ui.intro, org.eclipse.ui.intro.universal	Will later be used to implement the “Welcome” screen.
org.csstudio.startup	Provides basic CSS-compliant application code.
org.csstudio.utility.product	Provides extensions to the basic CSS-compliant application code.

Fundamentally, the application code of an RCP product is allowed to do pretty much anything. CSS end-user products, however, are expected to have a main window, a menu bar as described in Chapter 24, *CSS menus - org.csstudio.ui.menu*, and support for online help. A CSS product should support opening documents from the command line, see Chapter 27, *Opening Files from Command-Line - org.csstudio.openfile*. To simplify the creation of a compliant product, the `org.csstudio.startup` plugin provides the essential application code and extension points for customizing it according to local needs. Commonly used implementations of these extension points in turn are provided by the plugin `org.csstudio.utility.product`. Using these two plugins, a site-specific product can often be created without having to implement any application code.

Implement Application

In the new product plugin, extend the `org.eclipse.core.runtime.applications` extension point. As an ID, you can simply enter “application” which will result in a complete ID of `org.csstudio.mysite.product.application`. Add a “run” element with value `org.csstudio.startup.application.Application`, i.e. use the skeleton implementation from the CSS startup plugin.

The plugin `org.csstudio.startup` defines an extension point for customizing the behavior of its application code. In the new product plugin, extend this point which is called `org.csstudio.startup.module` and add parameters to it so that the corresponding section of your `plugin.xml` file looks as follows:

```
<extension point="org.csstudio.startup.module">
  <startupParameters
    class="org.csstudio.utility.product.StartupParameters">
  </startupParameters>
  <project
    class="org.csstudio.startup.module.defaults.DefaultProject">
  </project>
  <workbench
    class="org.csstudio.utility.product.Workbench">
  </workbench>
</extension>
```

The startup parameters code will parse command-line parameters. The default project code asserts that your product has at least one “CSS” project in its workspace. The workbench code, finally, is the most important part: It opens the window, configures it, and executes the main loop of the RCP application.

Add Product Definition

Use the IDE wizard to create a new “Plug-in Development”, “Product Configuration” in your product plugin. When editing the generated `.product` file, one of the first things you can set in the “General Information” section of the “Overview” tab of the product editor is the Name of your product. You should use “Css” as the name, with a capital C exactly as in Css. (For an explanation see Chapter 27, *Opening Files from Command-Line - org.csstudio.openfile*).

Select the application ID defined in the previous step. Press the “New...” button next to the “Product Definition” section to create a new product ID, and in there again select your application ID.

Next you need to select if your product configuration is based on plugins or features, and then list either plugin or features in the “Dependencies” tab. Initially it is easier to base a product on plugins. You can list the application plugins that you want to include in your product, for example Probe and BOY. You then press the “Add Required Plug-ins” button, and you are done.

This approach has the disadvantage that your product consists of a long list of plugins. After the fact it will be hard to determine which plugins you wanted to include in your product to provide the user with some functionality, and which plugins had to be added because they were dependencies of those functional plugins. Also remember that you might need plugins that do not show up as direct dependencies.

In the long run your product will be easier to maintain if it is based on features. You define one feature to list the application plugins that you want for your users, and separate features to list the plugins that result from dependencies, maybe further separated into CSS core plugins and those from Eclipse. The following describes how to create those features.

Create `applications` Feature

Create a new “Feature” project called `org.csstudio.mysite.applications.feature`. This feature lists all the application plugins that you want to include into your product, for example the plugins for Probe, Data Browser, BOY. Maybe just Probe as you get started.

Start by adding your product plugin, then add the application plugins. Some application plugins are already pre-aggregated into features, for example `org.csstudio.opibuilder.feature`. You can add individual plugins or make your applications feature include other features.

Add the `applications` feature as a dependency to your `*.product`.

Create `core` Feature

Similar to the `eclipse` feature, create a feature called `org.csstudio.mysite.core.feature` and add it as a dependency to your product. This feature will list all “core” CSS plugins that your application plugins need. By separating these supporting plugins from the application plugins that the end user sees, it will be easier to maintain your product in the long run.

Some plugin that you will have to add:

- `org.csstudio.startup` - Required by our product
- `org.csstudio.utility.product` - ditto
- `org.csstudio.ui.menu` - Define CSS menu structure
- `org.csstudio.ui.help` - Define CSS help structure

We will soon add more core plugins. Add the `core` feature as a dependency to your `*.product`.

Create `eclipse` Feature

Finally, create an `eclipse` feature. This feature will list all Eclipse plugins, i.e. plugins that CSS uses but which are provided by Eclipse. It is unfortunately not very easy to determine which Eclipse plugins you need to include in your product, and details will change between versions of Eclipse. For this reason it is useful to list them in their own feature.

Start by adding `org.eclipse.core.runtime` to your `eclipse` feature, then add the `eclipse` feature as a dependency to your `*.product`.

Fix Dependencies

When a product is based on features, some manual labor will be required to add direct plugin dependencies to the `core` respectively `eclipse` features.

This will be painful. If you base your product on plugins, Eclipse can add all required plugins via a simple click of a button. But when you base your product on features, Eclipse cannot tell to which of the features the missing plugins should be added. You will have to do this. Still, in the long run your product should be easier to maintain if it is based on features, so hang in there.

Start by trying to run the new product from within the IDE: Open the `*.product` file, press “Synchronize” and then “Launch an Eclipse Application”. Your product will *not* start. Instead, you will see many error messages, including “The application could not start. Would you like to view the log?”. You can select to see the errors in the `Error log view`, which is typically more convenient, or you can read the plain log file that looks like this:

```
!MESSAGE Bundle .../org.csstudio.ui.help/ was not resolved.
!MESSAGE Missing required bundle org.eclipse.help.ui_0.0.0
```

This means that the plugin `org.eclipse.help.ui` is a missing direct dependency of your product. Add all missing plugins with names starting in org.csstudio into your `core` feature, and add the missing Eclipse plugins into your `eclipse` feature.

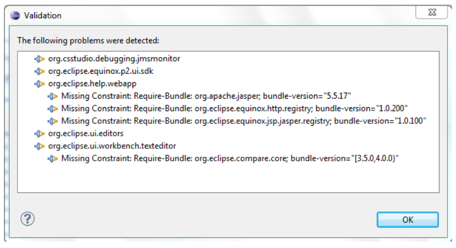
For some missing dependencies you will find that they are available in system-specific variants. One example is file system access. Your product might require a plugin `org.eclipse.core.filesystem`. When you add it to your eclipse feature, you will notice that there are similarly named plugins `org.eclipse.core.filesystem.win32.x86` and `org.eclipse.core.filesystem.macosx` because details of file systems differ between operating systems.

You should go ahead and add all variants, because eventually you want to build your product for multiple architectures. When doing this, however, you should configure the feature to only include those plugins on the appropriate target architecture. In the feature editor, you can enter the operating systems and architecture for each plugin. In the generated `feature.xml` file, it should look like this:

```
<plugin
   id="org.eclipse.core.filesystem.win32.x86"
   os="win32"
   arch="x86"
   ...
```

This way your product will include the Windows-specific file system support for x86 architectures, but only when your product is actually compiled for that target architecture.

Figure 15.2. Plug-in Dependency Validator


Validation

The following problems were detected:

- org.csstudio.debugging.jmsmonitor
- org.eclipse.equinox.p2.ui.sdk
- org.eclipse.help.webapp
  - Missing Constraint: Require-Bundle: org.apache.jasper; bundle-version="5.5.17"
  - Missing Constraint: Require-Bundle: org.eclipse.equinox.http.registry; bundle-version="1.0.200"
  - Missing Constraint: Require-Bundle: org.eclipse.equinox.jsp.jasper.registry; bundle-version="1.0.100"
- org.eclipse.ui.editors
- org.eclipse.ui.workbench.texteditor
  - Missing Constraint: Require-Bundle: org.eclipse.compare.core; bundle-version="[3.5.0,4.0.0)"

Another display of missing plugin dependencies is available from the menu “Run”, “Run Configurations...”. Locate the Eclipse Application entry for your product, select the “Plug-ins” tab and press the “Validate Plug-Ins” button. It will open a display of missing dependencies as shown in Figure 15.2, “Plug-in Dependency Validator”.

Continue to add missing dependencies to your features, occasionally pressing “Synchronize” and then “Launch an Eclipse Application” in your product to see which dependencies are still required.

Eventually, your product should run.

Congratulations! You just managed to overcome the most difficult part of getting started with a site-specific setup of CSS.

When you now add more plugins to your product, it is usually much easier to identify the required plugins because they will be direct dependencies of the added plugins. After a change in the Eclipse version, modifications to your product will mostly be limited to the eclipse feature.

Memory Settings

Java code always runs within a limited memory environment. The JVM emposes an upper limit on the amount of memory that it requests from the operating system. This means that a Java program is very unlikely to exhaust all the computer's memory and have a negative impact on other applications running on the same machine. On the other hand, if your product contains enough plugins or is used in a way that requires a lot of memory, your log file (see Section 5.2, “Log File”) can indicate out-of-memory errors because the JVM hit its self-imposed limit. Portions of CSS will stop functioning as expected: Display stops updating, new windows fail to open. The CSS product for your site should therefore be configured to allow for sufficient memory for its expected use, while at the same time not consuming all the computer's memory.

For Eclipse RCP products, these limits can be configured in the product file:

1. Open the `*.product` file
2. In the product configuration editor, select the “Launching” tab.
3. Enter JVM memory settings in the “VM Arguments” field.

The following will allow the JVM to fetch up to 1GB of dynamic memory and 128MB of code space:

```
-Xmx1024m
-XX:MaxPermSize=128M
```

The settings it the `*.product` file take effect the next time you build the product binary. If you run the product within the IDE during development and testing, you need to update the memory settings in the run configuration of the product. To adjust the settings of an existing product binary, you can add or edit the same commands in the `*.ini` of the product. The ini file has the same name as the application launcher, but with “.ini” as an extension. For Mac OS X, is is located in the `*.app/Contents/MacOS` folder of the product binary.

Unfortunately, it is hard to predict the best memory setting. You will have to try your product for a while in common use cases to determine how much memory you should allow.

Chapter 16. Product Intro Pages

16.1. “Welcome” Pages

Eclipse provides introduction pages that are accessible via the menu `Help`, `Welcome`. They are also displayed to the user when CSS is started for the very first time.

16.2. Universal Intro

Fundamentally, an RCP product can provide arbitrary introduction page content via the `org.eclipse.ui.intro` extension point. Refer to the online help for details.

CSS products, however, should use the “Universal Intro” mechanism. Instead of defining the complete introduction page content within your product, it allows other CSS plugins to also contribute intro page content. There is no need to update a custom, product-specific intro when adding or changing plugins.

To enable the Universal Intro, configure the intro extension point as follows for your product, using the correct `productId` of your product:

```
<extension point="org.eclipse.ui.intro">
 <introProductBinding
       introId="org.eclipse.ui.intro.universal"
       productId="org.csstudio.....your-product.....product"/>
</extension>
```

Your product's `plugin.xml` file should already contain an entry for the `org.eclipse.core.runtime.products` extension point. You need to add the following elements to configure the main page of the universal intro:

```
<extension point="org.eclipse.core.runtime.products">
  id="product"
  <product application="...." name="...">
    <property name="aboutImage" value="..."/>
    <!-- Append intro properties: -->
    <property name="introBrandingImageText"
       value="My Version of CSS"/>
    <property name="introBrandingImage"
       value="product:icons/css64.png"/>
    <property name="introTitle"
       value="Welcome to Control System Studio (CSS) for this site!">
    </property>
  </product>
</extension>
```

You can influence the behavior of the Universal Intro plugin to some extend via preference settings added to your product's `plugin_customization.ini` file, for example like this:

```
# Select sections of intro page
org.eclipse.ui.intro.universal/INTRO_ROOT_PAGES=overview,\
     firststeps,whatsnew

# Select scheme
org.eclipse.ui.intro/INTRO_THEME=org.eclipse.ui.intro.universal.slate
```

```
# Configure placement of sections
org.eclipse.ui.intro.universal/INTRO_DATA=product:intro_data.xml
```

The `intro_data.xml` file mentioned in these settings can look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This file controls the initial placement of
     Welcome (intro) page items.
     File can be generated via the Welcome page preference panel.

     Items that are not listed in here will show in a
     default place on their page.
     Items listen in <hidden></hidden> sections will be, well,
     hidden.
  -->
<extensions>
 <page id="overview">
  <group path="page-content/top-left">
   <extension id="org.csstudio.myproduct.product.overview"
              importance="high"/>
  </group>
 </page>
 <page id="whatsnew">
  <group path="page-content/top-left">
    <extension id="org.csstudio.sns.product.whatsnew"
               importance="high"/>
  </group>
 </page>
 <page id="firststeps">
  <group path="page-content/top-left">
    <extension id="org.csstudio.trends.databrowser"
               importance="medium"/>
  </group>
  <group path="page-content/top-right">
    <extension id="org.csstudio.opibuilder" importance="medium"/>
  </group>
 </page>
</extensions>
```

16.3. How to Contribute

The following mark-up in your plugin.xml file adds a link to your file `doc/overview.xml` to the intro pages:

```
<extension
    point="org.eclipse.ui.intro.configExtension">
 <configExtension
   configId="org.eclipse.ui.intro.universalConfig"
   content="doc/overview.xml">
 </configExtension>
</extension>
```

The IDE can help with this step: When adding a `..configExtension` to your product in the plugin editor, the "Universal Welcome Contribution" is listed as a template. It will generate an example intro file. Consult the Eclipse online help or example files from other CSS plugin for details on the intro file format.

Assuming you have a plugin `org.csstudio.XXX` and you want to provide intro content, there are two options for the location of that content:

- Within the same plugin `org.csstudio.XXX`.
- In a separate plugin `org.csstudio.XXX.intro`.

Using the same plugin is the easiest option. All users of the plugin will get the intro content. If site integrators want to disable it, they can do this via a `<hidden>` section in the `org.eclipse.ui.intro.universal/INTRO_DATA`. It does, however, add a dependency to the intro plugins to your plugin.

Using a separate plugin for the intoduction makes the inclusion or exclusion of the intro content more straight forward, but adds more plugins, and is additional work. It would allow building products without universal intro or even no intro at all. In the long run, intro content in a separate plugin is probably most flexible. In the short run, it might be OK to add the intro to the existing plugin, then extract it to a separate plugin as the need arises.

16.4. Where to Contribute

Technically, one can contribute to universal intro pages called `overview`, `firststeps`, `tutorials`, `samples`, `whatsnew`, `migrate`, `webresources`, or even define new pages.

For CSS products it is suggested to concentrate on the following:

`overview`
: This is where each site can publish their site-specific content:

```
Welcome to CSS for ... users!
CSS is ...
To get started, read the 'first steps'
that introduce the various tools.
You can also read the online help, ...
```

`firststeps`
: This is where each tool, via a separate `org.csstudio.....intro` plugin, can contribute some how-to-get-started content.

`whatsnew`
: This is were each site and each tool may add info about most recent changes, so users who've used CSS before can read up on what changed.

16.5. Issues

Main Intro Screen

The layout of the Universal Intro main screen is mostly fixed. There isn't much to customize beyond a one-line title and a small icon.

Cheat Sheets

In principle, it would be nice to be able to link to Cheat Sheets from for example the `firststeps` section with code like this:

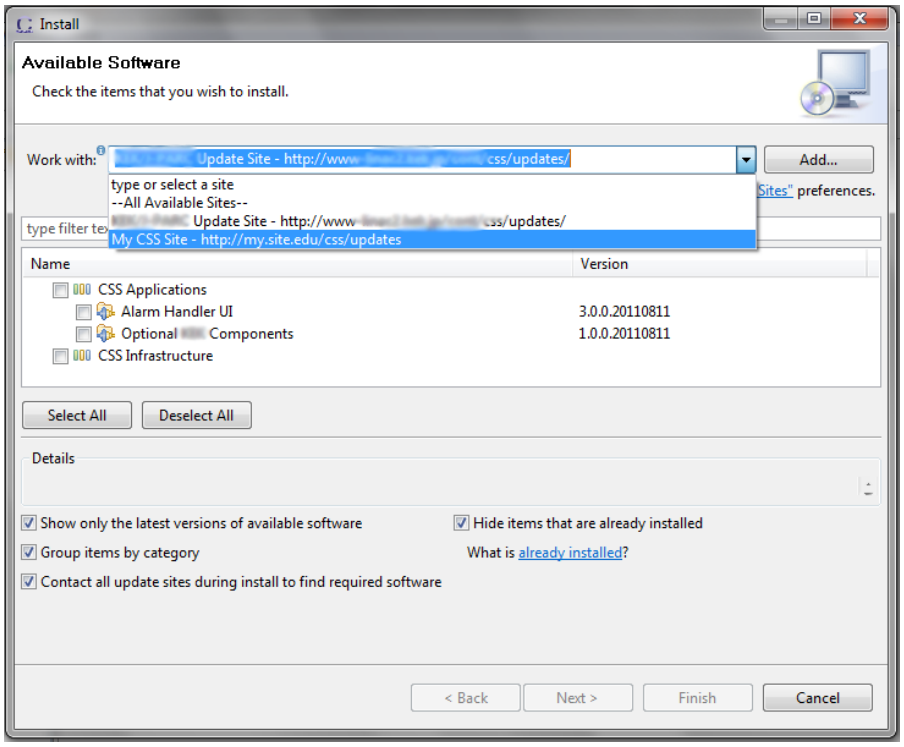
```
<!-- url is one string, broken for readability -->
<link
  url="http://org.eclipse.ui.intro/showStandby?
  partId=org.eclipse.platform.cheatsheet&amp;
  input=org.csstudio.XXXX.some_cheat_sheet"
```

But that requires the plugin `org.eclipse.platform`, which in turn adds quite some intro content that is useful for Eclipse programmers but potentially confusing for CSS end-users.

Chapter 17. Update Repository

The update repository is part of the P2 provisioning system. It allows users to install additional components into CSS. For example, the basic CSS product for your site might not include the alarm system user interface. Interested users can add it to CSS via the menu `Help`, `Install New Software...`

Figure 17.1. Installing from an Update Site



Install

Available Software

Check the items that you wish to install.

Work with: [redacted] Update Site - http://www[redacted]css/updates/

type or select a site
--All Available Sites--
[redacted] Update Site - http://www[redacted]css/updates/
My CSS Site - http://my.site.edu/css/updates

Name	Version
☐ CSS Applications	
☐ Alarm Handler UI	3.0.0.20110811
☐ Optional [redacted] Components	1.0.0.20110811
☐ CSS Infrastructure	

Details

☑ Show only the latest versions of available software
☑ Hide items that are already installed
☑ Group items by category
What is already installed?
☑ Contact all update sites during install to find required software

The update mechanism further supports automated updates of already installed software to a new version, for example by prompting users if they want to update whenever a new version becomes available.

Products that use P2 maintain their installation state. P2 tracks all the plugins and their versions that constitute the product. The product can be updated to a newer version from an update site, and in case of problems a product configuration can also be reverted back to a previous state.

We will briefly describe its usage, then how to create the necessary update site.

17.1. Usage

Usage of the update mechanism from the final product is relatively easy once the update site is in place. Most of the time, users to nothing. If an update becomes available, a small notification “Updates available. ... Click here to install” will appear the next time CSS is started. Users follow the sequence of dialogs, finally CSS restarts with the new updates in place.

A menu entry `Help`, `Check for Updates` can be used to trigger this process while Eclipse is already running.

Via the menu `Help`, `Install New Software...` users can install additional, optional software from the repository.

Finally, the menu `Help`, `About`, `Installation Details` provides access to the history of installed updates and allows you to revert to a previous state, i.e. to un-install software that has been added.

17.2. Create P2 Update Repository

When you export a product from the IDE as described in Section 4.5, “Product Export from IDE”, or when you perform a headless build of a product as per Section 4.7, “Headless Build”, you can select to not only build the product but also generate an update repository.

In the IDE export, check the option “Generate metadata repository”. Eclipse will export the product and in addition create a directory called `repository`.

In the headless build, set

```
p2.gathering=true
```

in your `build.properties` file, and Eclipse will create a directory `buildRepo`.

The update repository contains files `content.xml` and `artifacts.xml` (or jar file equivalents) as well as sub-directories with features, plugins and binaries.

By publishing this directory structure on a web server, P2-enabled products can then use the URL to that directory for online updates.

17.3. Enabling P2 Updates in a Product

To enable an RCP product to use an update site, you need to do this:

1. Include the P2 client tools.

   By including the feature

   ```
   org.eclipse.equinox.p2.user.ui
   ```

   in your product, you have the complete P2 GUI to support for example the menu `Help`, `Check for Updates` and `Help`, `Install new Software...`
2. Configure for your update site.

   Ideally, your product is already configured to look for updates on your update site, and maybe even performs this automatically with little user interaction.

   To include your update site in the configuration of your product, create a file `p2.inf` in your product plugin. The build tools will use this to configure the product at the end of the build:

   ```
   # Define P2 repository, see
   # http://wiki.eclipse.org/Equinox/p2/Engine/Touchpoint_Instructions
   # http://www.ralfebert.de/blog/eclipsercp/p2_updates_tutorial/
   instructions.configure=\
    addRepository(type:0,name: My Update Site,\
     location:http${#58}//www.my.site/css/updates/);\
    addRepository(type:1,name: My Update Site,\
     location:http${#58}//www.my.site/css/updates/);
   ```

   If you want your product to check for updates on each startup, prompting the user if she would like to perform the update, add this to the `plugin_customization.ini` of your product:

   ```
   ## P2 Updates: Enable check on startup.
   ```

```
# Only check, don't download
org.eclipse.equinox.p2.ui.sdk.scheduler/enabled=true
org.eclipse.equinox.p2.ui.sdk.scheduler/schedule=on-startup
org.eclipse.equinox.p2.ui.sdk.scheduler/download=false
```

17.4. Version Numbers

When you use online updates of our product, you need to pay attention to the version numbers of your plugins. When you change a plugin, you need to increment its version number so that P2 can detect a new version and support an update to that new version.

It is a good idea to use version numbers that end in `.qualifier` like

```
1.2.3.qualifier
```

The headless build will replace that qualifier with the current date and time, guaranteeing that the plugin has a unique version number even if you occasionally forget to increment the version number.

The headless build default for the qualifier replacement, i.e. date and time, can be a problem when you run a scripted build of several products. Your products will likely share some core CSS plugins, say `org.csstudio.logging`. With the default qualifier replacements, the headless build will create binaries like `org.csstudio.logging_3.0.0.v201108181010.jar`, `org.csstudio.logging_3.0.0.v201108181011.jar`, `org.csstudio.logging_3.0.0.v201108181012.jar`, that is different versions because the same plugin is re-built for each product with a slightly different time stamp.

A better solution might be to call your headless build with an option to defines a qualifier that only includes the current date, not time:

```
java -jar .../org.eclipse.equinox.launcher_*.jar \
  -application org.eclipse.ant.core.antRunner \
  ...
  -Dqualifier=`date "+%Y%m%d"` \
```

In your headless build configuration file, `build.properties`, you then use that variable as a qualifier:

```
# build.properties
product=...
...
forceContextQualifier=${qualifier}
...
```

17.5. Categories

When you review the user interface for installing additional features from a P2 repository, Figure 17.1, “Installing from an Update Site”, note the option to “Group items by category”. By default it will be selected, meaning that *only features that are categorized* will be displayed in the dialog.

When you try to use your update repository for the first time and wonder why you cannot find any of the expected content on your update site, try if de-selecting that option will reveal your updates. If it does, you are missing category definitions for your features.

To list your features in categories, create a file `category.xml` via `File`, `New`, `Plug-in Development`, `Category Definition` and list all your features.

In the headless build configuration for your products and features, reference that category file:

```
# build.properties
topLevelElementType=feature
topLevelElementId=org.csstudio....
...
# Use absolute path or maybe path relative to the builder directory
p2.category.definition=file:${builder}/../category.xml
...
```

17.6. Maintaining an Update Site

Your update repository can contain all the plugins for more than one version of a product. It can even contain the components for different products. This can be useful to allow updates of older versions of a product to either the latest or an intermediate version. You can update from say version 1.0 to 1.1, then to 1.2, then to 1.3, revert to 1.0, then directly update to 1.3.

To create a repository that contains multiple versions, you can simply leave an existing `repository` respectively `buildRepo` directory in place when you perform the new build. Eclipse P2 will add to the existing repository, it will not replace it.

You might, however, prefer to have better control over the content of your repository:

What if you want to reduce the size of your repository by deleting some older versions?

What happens to the repository when a build fails?

One suggestion is to always build a new repository via the headless build. Start with an empty repository, then copy the repository generated for a specific version into versioned subdirectories on the web server:

```

/path/to/web/root/css/updates/repo1.0
/path/to/web/root/css/updates/repo1.1
/path/to/web/root/css/updates/repo1.2
/path/to/web/root/css/updates/repo1.3
...
```

This way it is easy to delete the repository for a selected version. Next, combine those versions that you intend to expose to users with the P2 mirror application. A script similar to this will copy or add one repository to another:

```
# Mirror one P2 repository to another
# Mirror the metadata
java -jar $ECLIPSE/plugins/org.eclipse.equinox.launcher_*.jar \
  -application org.eclipse.equinox.p2.metadata.repository.mirrorApplication \
  -source  $SOURCE -destination $DEST
# Mirror the artifacts
java -jar $ECLIPSE/plugins/org.eclipse.equinox.launcher_*.jar \
  -application org.eclipse.equinox.p2.artifact.repository.mirrorApplication \
  -verbose \
  -compare \
  -source  $SOURCE -destination $DEST
```

Given such a mirror script, you can assemble a repository as desired:

```
cd /path/to/web/root/css/updates
# Delete old repository
```

```
rm -rf content.xml artifacts.xml binary features plugins
# Include versions 1.0 and 1.3 in combined repository
mirror.sh repo1.0 .
mirror.sh repo1.3 .
```

Chapter 18. Localization

Since control system tools are used in countries with different languages, it is often a good idea to localize the texts in CSS plugins. This way, they can be not only in English but also German, French, Chinese, Japanese or in other languages.

18.1. Externalize Source Code Strings

When developing source code for CSS, it is a good idea to enable compiler warning for non-localized strings, found in the Eclipse preferences under `Java`, `Compiler`, `Errors/Warnings`, `Code style` as “Non-externalized strings (missing/unused $NON-NLS$ tag)”.

With this warning enabled, it is easy to spot non-localized texts in Java source code. For example, the texts in the following will be marked as non-localized:

```
final String ID = "org.csstudio.someplug.someid";
label.setText("Hello");
```

To remove the warnings, there are fundamentally three options:

- Some strings like plugin IDs, host names, certain file names are not meant to be localized. The software depends on the fixed value of these strings. You can mark them with a NON-NLS comment like this to indicate that the text cannot be localized:

  ```
  final String ID =
     "org.csstudio.someplug.someid";  //$NON-NLS-1$
  ```

  The Eclipse Quick Fix in the context menu of the warning can add appropriate NON-NLS comments.
- Sometimes a whole method or class cannot be localized because all the texts in the are internal to the application and not meant to display in a user interface. In that case, adding

  ```
  @SuppressWarnings("nls")
  ```

  to the method or class will suppress all localization warnings.
- Finally, if the text should indeed be localized, it is best to use the tool invoked from the menu `Source`, `Externalize Strings...` to extract the texts into a separate `Messages` class. Your code will then look like this:

  ```

  label.setText(Messages.Hello);
  ```

  The generated `Messages` class will load the text from a properties file.

18.2. Message Properties Files

The `Externalize Strings...` tool will create a file `messages.properties` that includes the extracted test, for example:

```
# File messages.properties
Hello=Hello
```

This file will be used as a default. You create translations by adding additional property files:

```
# File messages_de.properties
Hello=Hallo
```

```
# File messages_fr.properties
```

```
Hello=Bonjour
```

Special characters like the German u-umlaut need to be written in unicode as `\u00fc`

18.3. Test Localization in IDE

To run CSS with a different localization, you typically need to install it on a computer with an operating system that uses that localization. For example, on a computer with the German version of Windows, CSS will use the `de` localization files.

It is also possible to pass a command-line argument to the Eclipse launcher to force a specific localization. Products launched within the IDE will typically include this as a default program argument:

```
-nl ${target.nl}
```

By changing the run configuration to use

```
-nl de_DE
```

you can run the product with German localization.

18.4. Language Codes

The locale identifier consists of a language and country. For the `messages.properties` file names, it is often sufficient to only use the language code as in `messages_fr.properties` for French, but in principle you could also create files `messages_fr_FR.properties` and `messages_fr_CA.properties` with different localizations for France and Canada.

Other example language and country codes:

- en_US: English, USA
- en_CA: English, Canada
- de_DE: German, Germany
- zh_CN: Simplified Chinese, China
- ja_JP: Japanese, Japan

18.5. Externalize Texts in plugin.xml

To localize for example the labels of menu items or the title of a view, you need to localize the corresponding text in the `plugin.xml`:

1. Create a file `plugin.properties` with content like this:

   ```
   TryThis=Try this!
   ```
2. Add this option to the `MANIFEST.MF` which instructs Eclipse to use the properties file:

   ```
   Bundle-Localization: plugin
   ```
3. Replace fixed texts in the plugin markup with references to the texts defined in the properties file using a percent sign:

   ```
   <page name="%TryThis">
   ...
   ```
4. Create translated files like `plugin_de.properties`

18.6. Language Caveats

It is often not sufficient to simply translate words, because the structure of sentences can be very different in another language. For example, when a message needs to be constructed based on PV names and an error, it is suggested to use the `NLS` formatting utility for this:

```
final String pv = .. some PV name;
final String error = .. some error info;
final String message =
  NLS.bind("There was an error {0} with PV {1}",
           error, pv);
```

This way, the message format can be externalized:

```
NLS.bind(Messages.PVErrorFmt, error, pv);
```

By default, the `PVErrorFmt` would be “There was an error {0} with PV {1}”, but it can be translated into another language with a different sentence structure, for instance “The PV {1} produced an error. Original error description: {0}.”

18.7. Externalize Texts in Online Help

To localize online help, the default online help subdirectory needs to be replicated into subfolders `nl/de`, `nl/zh` and so on for the various languages. Refer to eclipse online help for details.

Chapter 19. Access to Data

CSS code uses “plug-able” libraries for accessing data.

19.1. Live Data

There are at this time three libraries for accessing live data:

1. DAL: This library was developed to support all data types, narrow PV-type access as well as wide object-type access, for EPICS as well as other control system network protocols. It is used by SDS.
2. utility.pv: This smaller library was developed to support PV-type access for the data types needed by generic control system tools. It is used by the EPICS PV Tree, PV Table, Data Browser, BOY, (BEAUTY) Archive Engine, (BEAST) Alarm Server.
3. pvmanager: This new API supports PV-type access but also handles aggregation and threading.

All libraries are fundamentally plug-able as described in the following where utility.pv is used as an example.

The plugin `org.csstudio.utility.pv` defines an interface for accessing live control system data. The archive engine for example uses that library for subscribing to value updates from PVs that you want to archive. The utility.pv plugin does not, however, contain any implementation. Instead, it defines an Eclipse Extension Point that allows other plugins to provide pluggable implementations. The plugin `org.csstudio.utility.pv.epics` implements live data access based on EPICS Channel Access Version 3. The plugin `org.csstudio.utility.pv.simu` implements simulated PVs like `sim://ramp`. This way, one can build an archive engine that supports EPICS, or EPICS and simulated PVs, or only simulated PVs by simply including the desired plugins. There is no need to change the actual archive engine code at all.

While this modular approach is very flexible, there can be one disadvantage: The archive engine code for example only depends on `org.csstudio.utility.pv`, the definition of the API for accessing live data. When bundling the archive engine code into a product, this plugin must be included. The implementing plugins like `org.csstudio.utility.pv.epics` are optional, allowing you to build an archive engine that does not interface to EPICS but another network protocol of your choice. If you fail to include *any* implementing plugins, the product will built without errors but later issue runtime error messages

```
No extensions to org.csstudio.utility.pv.pvfactory found
```

This error means: The `org.csstudio.utility.pv` could not locate any implementation, no “factory” classes for creating actual live data PVs. You need to include at least one implementing plugin like `org.csstudio.utility.pv.epics` or `org.csstudio.utility.pv.simu`.

Chapter 20. Data Exchange within CSS

Maybe the biggest difference between arbitrary RCP plugins and CSS code is the use of common data types, allowing for exchange of these data types via context menus or drag-and-drop. This chapter explains some of the underlying details for those who want to implement their own plugin code that links to CSS.

20.1. CSS Data Types

The plugin `org.csstudio.csdata` defines control system data types like `ProcessVariable`, a class that holds the name of a PV. By using this data type, CSS code can distinguish PV names from arbitrary strings.

20.2. Context Menu Contributions

Applications can define context menus. For example, when the user right-clicks on the list of traces in the Data Browser configuration, a context menu appears that allows operations like adding a trace, removing the selected trace etc.

One very powerful aspect of RCP is the way it allows code to contribute to context menus of *other* application code. For example, the Data Browser configuration panel defines a context menu with entries for editing the configuration of the current data browser. For those data browser traces that are based on PVs, the context menu will include links to other CSS tools that are capable of handling PVs. The underlying mechanism works as follows.

Use or Adapt to CSS Types

The application has to provide data in the form of common CSS data types like `ProcessVariable`. It can do that by directly using these data types, but in reality the data model of an application probably needs to store additional information, for example a PV name with color and other attributes. In that case it needs to implement an Eclipse adapter via the extension point `org.eclipse.core.runtime.adapters` from its internal model data types to CSS data types like `ProcessVariable`.

Allow additions to the context menu

When defining a context menu, all RCP applications are encouraged to include one item named "additions" that can be used by other RCP plugins to extend the context menu:

```
MenuManager menu = .... my menu ...;
menu.add(new Separator(IWorkbenchActionConstants.MB_ADDITIONS));
```

Contributing to context menus

The plugin `org.csstudio.ui.menu` defines a context menu with ID `org.csstudio.ui.menu.popup.processvariable` that is automatically added to all context menus where the selection adapts to a CSS `ProcessVariable`. If you want your tool to appear in such menus, you need to hook into the PV context menu with mark-up similar to the following:

```
<!-- Your plugin.xml -->
<extension point="org.eclipse.ui.menus">
 <menuContribution
   locationURI="popup:org.csstudio.ui.menu.popup.processvariable">
   <command commandId="org.csstudio.my_app.OpenMyTool"
         icon="icons/my_app.gif"
         style="push">
    </command>
 </menuContribution>
</extension>
```

This defines a command for the PV context menu. The command is further linked to the actual implementation (handler):

```
<extension point="org.eclipse.ui.commands">
  <command id="org.csstudio.my_app.OpenMyTool"
    defaultHandler="org.csstudio.my_app.OpenMyTool">
  </command>
</extension>
```

Handling the invocation from a context menu

Finally, you implement the handler that will be invoked from the context menu like this to receive for example the PV names:

```

package org.csstudio.my_app;
public class OpenMyTool extends AbstractHandler
{
    @Override
    public Object execute(final ExecutionEvent event)
      throws ExecutionException
    {
        final ISelection selection =
         HandlerUtil.getActiveMenuSelection(event);
        final ProcessVariable[] pvs =
         AdapterUtil.convert(selection, ProcessVariable.class);
        // Open my view, display the PVs, ...
```

20.3. Drag-and-Drop

Eclipse provides Drag-and-Drop support for data types like text and file names. CSS adds Drag-and-Drop support for any data type that is `Serializable`. The data types from Section 20.1, “CSS Data Types” like `ProcessVariable` are already `Serializable`, so application code that uses the CSS data types or adapts to them can easily participate in Drag-and-Drop data exchange. Drag-and-Drop is also possible within an application via data types that are only used within that application as long as they are `Serializable`. As a minimum denominator, data can be exchanged as text, which can be useful when interfacing with non-CSS applications.

Drag Source, Drop Target

The plugin `org.csstudio.ui.util` offers helper classes `ControlSystemDragSource` and `ControlSystemDropTarget` that allow “dragging” respectively “dropping” of any data type that supports serialization:

```
ProcessVariable pv = new ProcessVariable("Fred");
// Assume view somehow displays the pv
TableView view = ...;
// Allow dragging the PV out of the view
```

```
new ControlSystemDragSource(view.getControl())
{
   public Object getSelection()
   {
       return pv;
   }
};
```

```
// Other control that should allow dropping a PV
Control ctl = ...;
new ControlSystemDropTarget(ctl, ProcessVariable.class, String.class)
{
  public void handleDrop(final Object item)
  {
      if (item instanceof ProcessVariable)
          ctl.setText(((ProcessVariable) item).getName());
      else
          ctl.setText((String) item);
  }
};
```

Common Pitfalls

When adding Drag-and-Drop support to an application plugin, it can be helpful to enable detailed logging. The warning “Serialization failed” because of a `NotSerializableException` indicates that the data passed to the `ControlSystemDragSource` is not fully serializable. Maybe the class itself was marked as implementing `Serializable`, but one or more member variables are using non-serializable objects.

If the “De-Serialization fails” because of a `ClassNotFoundException`, the reason could be that the class used for the transfer is not visible outside the plugin that defines it. The code that performs the data transfer is in the plugin `org.csstudio.ui.util`. It can only de-serialize data with known object types. If your plugin defines a new data type `MyDataType implements Serializable`, you also need to list its package name in the `Export-Package` section of your plugin `MANIFEST.MF`. In addition, the package name of your data type must start with the plugin name, because the package name is used to determine which plugin class loader to use to create the de-serialized data instance. Even if you only plan to drag-and-drop data between views within one and the same plugin, all custom data types used for those transfers must be visible to the plugin `org.csstudio.ui.util`.

When transferring array data types, care must be taken to provide instances of the actual data type for the array like

```
new MyDataType[] { new MyDataType(), new MyDataType() }
```

The transfer will fail when sending an object array, even if it contains only instances of the supported data type:

```
new Object[] { new MyDataType(), new MyDataType() }
```

When dragging the currently selected data out of a JFace `Viewer`, it may thus be necessary to convert the selection before passing it to the drag source:

```
new ControlSystemDragSource(viewer.getControl())
{
   public Object getSelection()
   {
     final IStructuredSelection selection =
      (IStructuredSelection) viewer.getSelection();
     final Object[] objs = selection.toArray();
     final ProcessVariable[] pvs = Arrays.copyOf(objs, objs.length,
                    ProcessVariable[].class);
     return pvs;
   }
};
```

Part II. Plug-in Reference

The following chapters contain references for certain CSS plugins.

Chapter 21. CSS Core - org.csstudio.core.feature

CSS Core is the group of plug-ins that every CSS distribution has, regardless of product or site requirements. This set is kept as small as possible, to avoid dependency creep-in. A plug-in is added to this group only if, after a discussion on the mailing list, a qualified majority emerges in favor of the inclusion.

The `org.csstudio.core.feature` is used to maintain the list of plug-ins that belong to this group.

Chapter 22. CSS Core Utilities - org.csstudio.core.util.feature

CSS Core Utilities is the group of plug-ins that are supported to be shared across different sites to provide commons functionality among CSS applications, and constitute most of CSS core infrastructure. This includes data definitions, connection services, common Eclipse RCP items or utilities. A plug-in can be added to this feature if the following conditions are met:

1. there is interested from more than one site in using this plug-in
2. the maintainer takes responsibility for support to other sites

The `org.csstudio.core.util.feature` is used to maintain the list of plug-ins that belong to this group, and is not mandated to be packaged and distributed in a product (though one may decide to do so).

Chapter 23. Logging - org.csstudio.logging

CSS application code might need to log messages about warnings, fatal errors, but also informational messages. The suggestion is to use `java.util.logging`, the logging package that is included with Java.

23.1. Write Log Messages

To write log messages from application code, no additional CSS plugin is need. Simply invoke the logging API like this:

```
// Import logger from JRE
import java.util.logging.Logger;

// Fetch Logger, for example using current class name or plugin ID.
Logger logger = Logger.getLogger(getClass().getName());

// Log a messages
logger.warning("Something terrible happened");
logger.info("FYI, I just did something");

// Can use a formatter for lazy message generation
logger.log(Level.DEBUG, "Value is {0}", value);

// ... or to include detail of an exception
catch (Exception ex)
{
 logger.log(Level.WARNNIG, "Operation failed", ex);
}
```

23.2. Configure the Log System

There are several ways to configure `java.util.logging`, for example via *.ini files in the JRE. CSS includes a plugin `org.csstudio.logging` that supports logging in several ways:

- Configure logging based on Eclipse preferences. This way, you can configure logging together with other CSS plugins, see Chapter 6, *Hierarchical Preferences*.
- Send log messages to the Eclipse Console View in addition to the standard output (terminal window).
- Send log messages to files, allowing rotation between several files.
- Send log messages to JMS, which allows the collection of log messages from several sources.

To use `java.util.logging`, your product needs to invoke

```
LogConfigurator.configureFromPreferences()
```

from within its startup code, usually just before entering the Workbench run loop. The LogConfigurator registers a `PluginLogListener` to add RCP log messages to `java.util.logging`. Then it reads Eclipse preferences to configure logging, allowing to log to the console, files and JMS.

For details on how the logging to the console, files and JMS can be configured, refer to the file `org.csstudio.logging/preferences.ini`

A related plugin `org.csstudio.logging.ui` allows adjustments of the log preferences from the preferences GUI.

23.3. Logging to other systems

There are several other logging systems for Java: Log4j, Apache Commons Logging, SLF4J, ... The point for shared CSS code should be to not force the use of any particular external logging library into a CSS product. Shared CSS code should be content with the `java.util.logging` package provided by the standard Java library.

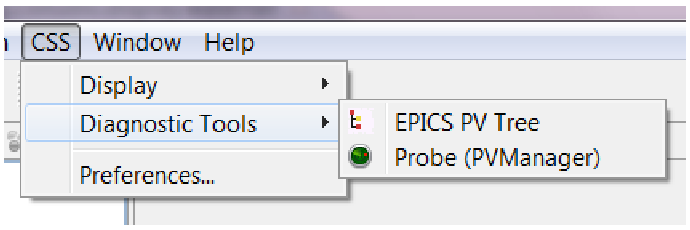
When creating a site-specific product, you are of course free to include for example SLF4J, and use its “bridge” as a root logger for java.util.logging, so all CSS log messages will then be piped through SLF4J.

Chapter 24. CSS menus - org.csstudio.ui.menu

The plugin “org.csstudio.ui.menu” defines through extensions the CSS Menu (which appears in the menu bar) and the context sensitive pop-up submenu for object adaptable to `org.csstudio.csdata.ProcessVariable`. The plugin “org.csstudio.ui.menu” provides examples of how to contribute actions/commands, how to implement a view that displays the pop-up menu, and with the same view one can test whether a popup action/commands actually displays as intended.

24.1. The CSS main menu

Figure 24.1. The CSS main menu



The overall structure of the menu is:

```
CSS (id: css)
 - Display (id: display)
 - Alarm (id: alarm)
 - Diagnostic (id: diag)
 - Debugging (id: debugging)
 - Configuration (id: configuration)
 - Management (id: management)
 - Editors (id: editors)
 - Utilities (id: utility)
 - Trends (id: trends)
 - Test (id: test)
 - Other (id: other)
```

where the each line has the name of each sub-menus and the “id” which is needed to define contributions to those menus. Only sub-menus that have contributions are going to be displayed

The following example adds a command to the `display` sub-menu:

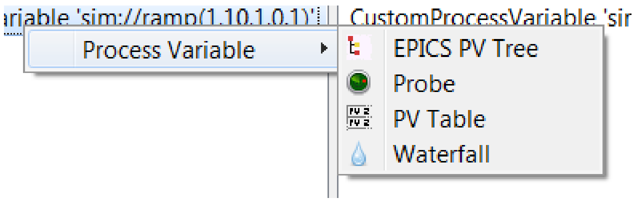
```
<plugin>
    ...
    <extension
         point="org.eclipse.ui.menus">
      ...
      <menuContribution
            allPopups="false"
            locationURI="menu:display">
         <command
               commandId="org.eclipse.ui.views.showView"
               icon="icons/my_icon.png"
               label="My View"
               style="push"
               tooltip="Show My View">
            <parameter
                  name="org.eclipse.ui.views.showView.viewId"
                  value="org.csstudio.my_tool.MyView">
            </parameter>
         </command>
      </menuContribution>
   </extension>
```

This example shows the rather common case where your menu entry opens a View, using the command `org.eclipse.ui.views.showView` that is provided by Eclipse. That command expects a command parameter to provide the ID of the view to open.

You can of course also add new commands to the menus. In that case, you also need to define a handler for your command. Refer to the Eclipse online help on commands and handlers for details.

24.2. The Process Variable popup-menu

Figure 24.2. The Process Variable popup-menu



This popup menu (id: `org.csstudio.ui.menu.popup.processvariable`) will appear only on objects that are adaptable to `org.csstudio.csdata.ProcessVariable`. Given that the whole menu is hidden/displayed, one only has to contribute commands without worrying about the display condition for each command. It also makes it easier for the user to predict which entries he will find in the menu and where they are located: if the “Process Variable” sub-menu is present, then the object is of the right type and all the same commands are going to be available in the same order.

The following example adds a command to the popup menu:

```
<plugin>
 ...
 <extension
     point="org.eclipse.ui.menus">
  ...
  <menuContribution
   allPopups="false"
   locationURI="popup:org.csstudio.ui.menu.popup.processvariable">
   <command
    commandId="org.csstudio.display.waterfall.SomeCommand"
    icon="icons/water.png"
    style="push">
   </command>
  </menuContribution>
  ...
 </extension>
 ...
</plugin>
```

To handle the received PV names in a command handler, see Chapter 26, *Common SWT/JFace utilities - org.csstudio.ui.util*

The following example adds an action to the popup menu (note the contribution to the `main` section):

```
<plugin>
...
<extension
    point="org.eclipse.ui.popupMenus">
 ...
 <objectContribution
   adaptable="true"
   id="org.csstudio.ui.menu.test.objectContribution1"
   objectClass="org.csstudio.csdata.ProcessVariable">
   <action
     class="org.csstudio.ui.menu.test.TestPVAction"
     icon="icons/test.png"
     id="org.csstudio.ui.menu.test.testpvaction"
     label="Test Action"
     menubarPath=
       "org.csstudio.ui.menu.popup.processvariable/main">
   </action>
 </objectContribution>
 ...
</extension>
...
</plugin>
```

Chapter 25. PV Access - org.csstudio.utility.pv

This plugin provides one of the CSS APIs for accessing live control system data, i.e. read process variable samples from front-end computers. Its emphasis is on fairly straight-forward access to individual PVs, using a listener for received updates.

25.1. Usage

The `org.csstudio.utility.pv` plugin defines the API for accessing PVs, an extension point to implement such PVs, and a `PVFactory` to access available implementations.

Typical usage looks like this:

```
// Create PV
final PV pv = PVFactory.createPV(pv_name);
// Register listener for updates
pv.addListener(new PVListener()
{
    public void pvDisconnected(PV pv)
    {
        System.out.println(pv.getName() + " is disconnected");
    }
    public void pvValueUpdate(PV pv)
    {
        IValue value = pv.getValue();
        System.out.println(pv.getName() + " = " + value);
        if (value instanceof IDoubleValue)
        {
            IDoubleValue dbl = (IDoubleValue) value;
            System.out.println(dbl.getValue());
        }
        // ... or use ValueUtil
    }
});
// Start the PV
pv.start();

...

pv.stop();
```

The `IValue`-derived data objects contain not only a basic value, i.e. number or string, but also a time stamp, status/severity information and in most cases meta data. The meta data provides information for display tools: value range, alarm limits, units, or strings that represent the states of enumerated PVs.

25.2. Available Implementations

The plugin `org.csstudio.utility.pv` defines an interface for accessing live control system data. The archive engine for example uses that library for subscribing to value updates from PVs that you want to archive.

The `utility.pv` plugin does not, however, contain any implementation. Instead, it defines an Eclipse Extension Point that allows other plugins to provide pluggable implementations.

By including either one or all of the following plugins in a product like the archive engine, one can build an archive engine that supports EPICS, or EPICS and simulated PVs, or only simulated PVs.

To associate PVs with an implementation, a prefix is used, for example `sim://sine` for a simulated PV “sine” or `ca://fred` for a Channel Access PV “fred”. The utility.pv plugin has a preference setting to specify a default prefix, which is usually set to the control system protocol. In that case, simply specifying a PV name of `fred` will also select `ca://fred`.

EPICS Channel Access

The plugin `org.csstudio.utility.pv.epics` implements live data access based on EPICS Channel Access Version 3.

Note that the Channel Access settings like the CA server address list are configured from Eclipse properties defined in the plugin `org.csstudio.platform.libs.epics`.

One important setting is the choice of “pure Java” or the JNI implementation of JCA. When selecting the former, the CAJ library will be used. CAJ is a pure Java Channel Access client implementation, usable on all platforms, and generally offering good performance. Choose JNI if you prefer to use the original Channel Access client library from EPICS base. The JNI client library guarantees full compatibility with existing EPICS infrastructure, but requires a system-specific version of the JNI binary. Refer to the `README.txt` in the libs.epics plugin for instructions on building this binary.

The libs.epics settings also include the subscription mode:

- VALUE - Subscribe to all value changes. Analog EPICS records can apply an MDEL update threshold. This is the default mode for CSS.
- ARCHIVE - Subscribe to archive updates. For analog EPICS records, this uses the dedicated ADEL update threshold. This should be used for the ArchiveEngine.
- ALARM - Subscribe to alarm state/severity changes. This should be used for the AlarmServer.

The utility.pv plugin attempts to read the `DBR_CTRL_...` meta data for each PV *once* on connection. EPICS IOCs will fill the meta data based on certain record fields like EGU, PREC, HIHI, depending on the record type.

The PV plugin then subscribes to the `DBR_TIME_...` type of the channel. This way the full meta data of the channel is known and can be returned with each value, while the network traffic for the subscription updates is reduced to the essential time, status and basic number or string. This procedure is quite common for EPICS client tool, but has the disadvantage that runtime changes to the meta data will not be noticed unless the client disconnects and then re-connects, since the full meta data is only obtained upon channel connection.

Simulated Data

The plugin `org.csstudio.utility.pv.simu` implements simulated PVs like

```
sim://ramp
```

One purpose of simulated PVs is for testing CSS tools in the absense of an actual control system. In an operational setup, “local” PVs like `loc://demo` can be useful to communicate via PVs inside CSS, for example between operator interface panels inside one instance of CSS.

Refer to the online help of the plugin `org.csstudio.utility.pvmanager.ui` for the syntax of simulated PVs.

25.3. Common Issues

Cannot read EPICS PVs

If you built CSS yourself from sources, assert that the EPICS plugins are included. Under `Help`, `About...`, `Installation Details`, `Plug-Ins` check that the plugins `org.csstudio.utility.pv.epics` and `org.csstudio.platform.libs.epics` are available.

Check the EPICS CA address list. Try to access the same PV with the EPICS `caget` command-line tool. Compare the EPICS_CA_ADDR_LIST environment variable used by the command-line tool with the CSS preference for the address list.

Try both the CAJ and JNI version of CA.

CA Repeater

Channel Access uses a `caRepeater` to monitor beacons from CA servers. If you run CSS without a CA repeater, you might see error messages

```
failed to start executable - "caRepeater" Cannot find the file
```

This error means: You have not started a CA repeater on the computer where you run CSS. In practice, CSS will work just fine most of the time. In case of network errors or IOC reboots, however, it might not re-connect to some PVs after the network or IOC problem should have been resolved until you re-open the affected tools.

In an operational control room setup, you should assert that all computers launch an instance of the EPICS caRepeater on bootup.

No Implementations in Product

While the modular plug-in approach is very flexible, there can be one disadvantage: The archive engine code for example only depends on `org.csstudio.utility.pv`, the *definition* of the API for accessing live data. When bundling the archive engine code into a product, this plugin must be included.

The *implementing* plugins like `org.csstudio.utility.pv.epics` are optional, allowing you to build an archive engine that does not interface to EPICS but another network protocol of your choice. If you fail to include *any* implementing plugins, the product will built without errors but later issue runtime error messages

```
No extensions to org.csstudio.utility.pv.pvfactory found
```

This error means: The `org.csstudio.utility.pv` could not locate any implementation, no factory classes for creating actual live data PVs. You need to include at least one implementing plugin like `org.csstudio.utility.pv.epics` or `org.csstudio.utility.pv.simu`.

Chapter 26. Common SWT/JFace utilities - org.csstudio.ui.util

The plugin “org.csstudio.ui.util” defines common ui elements can be used in different applications.

26.1. Adapter utilities - org.csstudio.ui.util.AdapterUtil

The most important way for Eclipse RCP plug-ins to communicate is throught the use of Adapters. Unfortunately Adapters do not work well with conversions from one object to arrays of an object of different kind, so we created a few utility methods to properly handle this case. These should be used when adapting the selection from one plug-in to the other during events like drag'n'drog and context menu command/actions.

For example, this command handler convert a selection to a specific type:

```
public class MyCommandHandler extends AbstractHandler
{
   @Override
   public Object execute(ExecutionEvent event)
      throws ExecutionException
   {
       ISelection selection =
          HandlerUtil.getActiveMenuSelection(event);
       ProcessVariable[] pvs =
          AdapterUtil.convert(selection, ProcessVariable.class);
    ...
}
```

If no selection is available, or no conversion is available, and empty array is returned. If each item in the selection is adaptable to a PV[], those arrays will be merged into a single array.

26.2. Drag and drop - org.csstudio.ui.util.dnd

This supports easier implementation for drag sources and drop targets.

To declare a source:

```
        Control control = ...

        // Drag PVs out of control
        new ControlSystemDragSource(control) {
            @Override
            public Object getSelection() {
                return pvs;
            }
        };
```

The source will take care of broadcasting the selection in all possible adaptable types.

To accept a drop:

```
        Control control = ...

        // Accept PVs for a drop
        new ControlSystemDropTarget(control,
            MyData.class,
            ProcessVariable[].class) {
            @Override
            public void handleDrop(Object item) {
                if (item instanceof ProcessVariable[]) {
                    control.setText(
                       Arrays.toString((ProcessVariable[]) item));
                }
                if (item instanceof MyData) {
                    control.setText(((MyData) item).getText());
                }
            }
        };
```

The target will take care of requesting the type request, in the order of preference given in the constructor.

Chapter 27. Opening Files from Command-Line - org.csstudio.openfile

27.1. Goal

While it is quite easy to open display or other CSS configuration files inside CSS, i.e. when CSS is already running, there is often also the need to open such files from *outside* of CSS. This is especially important for sites that are transitioning to CSS. If legacy control system tools can somehow open CSS displays from the command line, i.e. also from within shell scripts, the transition can be much smoother.

27.2. Eclipse Launcher

The Eclipse Launcher supports a command-line option

```
--launcher.openFile  some_file_name.ext
```

When this option is found, the launcher will send the `SWT.OpenDocuments` event to the RCP application. The launcher will also check for another copy of the RCP application. If it detects that another copy of the RCP application is already running, it will send the event to that application.

Fundamentally, this accomplishes the goal: CSS can receive the names of files that it should open from the command line. Duplicate instances of CSS are automatically prevented.

27.3. Product Name

To benefit from this Eclipse feature, several steps are necessary.

The launcher needs to detect another instance of CSS that might already be running, and it has to be able to do this on all supported operating systems. Eclipse depends on the following correlation between the product name and the launcher name:

- The launcher name has to be a lower-case name like `css`. You configure the launcher name in the “Launching” tab of the editor for your `*.product` file. On Windows, the resulting launcher is actually called `css.exe`.
- The application name must match the launcher name with the first letter capitalized, i.e. “Css”. You specify the application name on the “Overview” tab of the editor for your `*.product` file, in the “General Information” section. After pressing the “Synchronize” link on the Overview tab of the product editor, you should find the same application name in the product's `plugin.xml` file like this:

```
<extension id="product"
    point="org.eclipse.core.runtime.products">
 <product
    application="org.csstudio.your-site.product.application"
    name="Css">
    <property name="appName" value="Css">
    </property>
  ...
```

The key here is that the `appName` matches the name of your launcher with the first letter capitalized.

27.4. Handle `SWT.OpenDocuments`

Your CSS application needs to handle the received `SWT.OpenDocuments` events. When you use the `Workbench` class from the plugin `org.csstudio.utility.product`, its `ApplicationWorkbenchAdvisor` will attempt to open all received files. To accomplish this, it relies on the `DisplayUtil` from the `org.csstudio.openfile` plugin.

27.5. Associate File Types with Handlers

The `org.csstudio.openfile` plugin declares an extension point `org.csstudio.openfile.openDisplay` that other plugins can implement to support opening their files from the command-line.

One example is BOY, which will open `*.opi` files in the BOY runtime.

One might think that the existing Eclipse registry entries for editors are sufficient to associate file types with a handler that can open them. The editor registry, however, allows for the registration of multiple editors. When using that to open files from the command line, it would be hard to predict if for example an `*.opi` file was opened in the desired BOY runtime, or in the BOY editor, a generic XML file editor or a plain text file editor. The designated `...openDisplay` extension point avoids such ambiguities.

27.6. Default Command-Line Action

As described so far, a CSS product can open files from the command line when called like this:

```
css --launcher.openFile some_file.opi
```

When adding the following to the `*.ini` file of your product, simply listing the file name is sufficient:

```
--launcher.defaultAction
openFile
```

Note that this is the `*.ini` file of the product, not the `configuration/config.ini`. If the product is called `css`, that file is in the same directory as the launcher and called `css.ini`. Instead of manually editing the file, add the `--launcher.defaultAction openFile` in the `*.product` file under “Launching Arguments”, “Program Arguments”.

For one, this can be more convenient. In addition, this is necessary to support opening files on windows when you double click a file that is associated with CSS, or you select files and choose “Open With” or “Sent To” CSS, because that mechanism will invoke CSS with just the file names, lacking the `--launcher.openFile` option.

Chapter 28. Logbook Support - org.csstudio.logbook

Many sites have an electronic logbook, i.e. an online system that keeps entries with texts or images. CSS itself does not include an electronic logbook, but it can be linked to an existing site-specific system.

If a logbook is available at your site, several CSS tools will allow users to send text or screenshots to it. The Data Browser and BOY for example allow sending a screenshot to the logbook, while the alarm system tools can send a description of currently active alarms to the logbook. This can be done directly from for example the context menu of a BOY display, without need to start an external tool to create a screenshot, saving it to a file, then logging into the logbook to download the saved file etc.

28.1. org.csstudio.logbook

This plugin defines the API for making logbook entries as well as an extension point for the actual implementation.

Code that allows logbook entry generation depends on this plugin and can query it for the presence of an implementation. If no implementation of the logbook extension point is available, the logbook functionality should simply be disabled, for example by hiding corresponding menu entries.

28.2. org.csstudio.logbook.ui

This plugin provides a view for making simple text-based logbook entries.

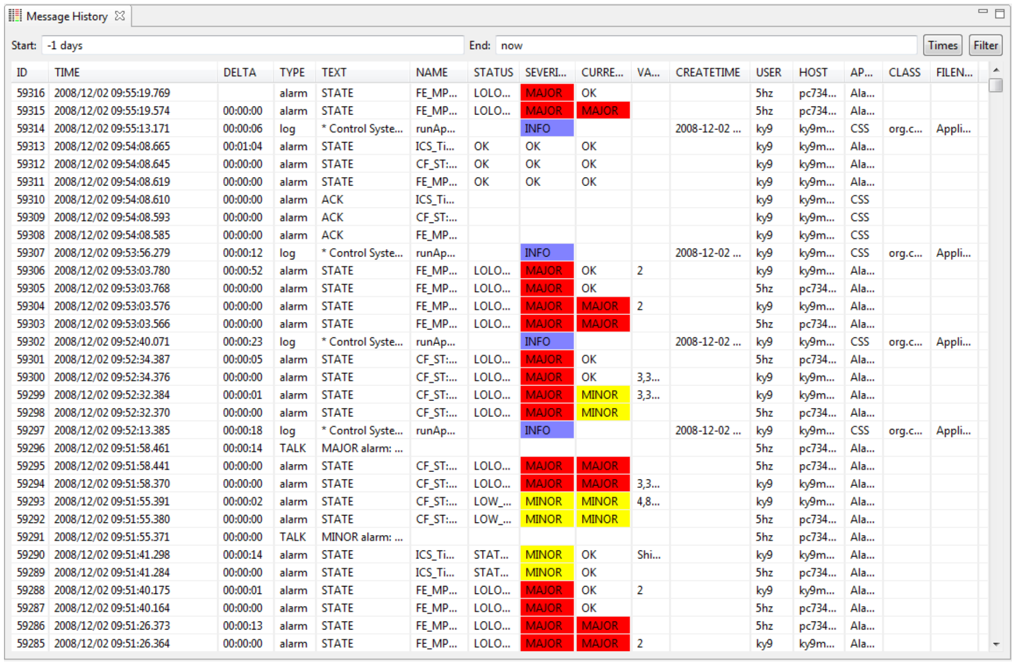
28.3. org.csstudio.logbook.sns, ...

Site-specific plugins provide the implementation for creating a logbook entry. There must be at most one such implementation. With no implementation available, the logbook functionality is simply disabled. When multiple implementations are available, the logbook support will report an error.

Chapter 29. Message History Browser - org.cstudio.alarm.beast.msghist

The plugin `org.cstudio.alarm.beast.msghist` is a generic browser for the message history. It displays JMS messages that were written to the relational database by for example the tool described in Chapter 33, *RDB Logging - org.cstudio.sns.jms2rdb*.

Figure 29.1. Message History Browser



Start: -1 days End: now

ID	TIME	DELTA	TYPE	TEXT	NAME	STATUS	SEVERI...	CURRE...	VA...	CREATETIME	USER	HOST	AP...	CLASS	FILEN...
59316	2008/12/02 09:55:19.769		alarm	STATE	FE_MP...	LOLO...	MAJOR	OK			5hz	pc734...	Ala...		
59315	2008/12/02 09:55:19.574	00:00:00	alarm	STATE	FE_MP...	LOLO...	MAJOR	MAJOR			5hz	pc734...	Ala...		
59314	2008/12/02 09:55:13.171	00:00:06	log	* Control Syste...	runAp...		INFO			2008-12-02 ...	ky9	ky9m...	CSS	org.c...	Appli...
59313	2008/12/02 09:54:08.665	00:01:04	alarm	STATE	ICS_Ti...	OK	OK	OK			ky9	ky9m...	Ala...		
59312	2008/12/02 09:54:08.645	00:00:00	alarm	STATE	CF_ST:...	OK	OK	OK			ky9	ky9m...	Ala...		
59311	2008/12/02 09:54:08.619	00:00:00	alarm	STATE	FE_MP...	OK	OK	OK			ky9	ky9m...	Ala...		
59310	2008/12/02 09:54:08.610	00:00:00	alarm	ACK	ICS_Ti...						ky9	ky9m...	CSS		
59309	2008/12/02 09:54:08.593	00:00:00	alarm	ACK	CF_ST:...						ky9	ky9m...	CSS		
59308	2008/12/02 09:54:08.585	00:00:00	alarm	ACK	FE_MP...						ky9	ky9m...	CSS		
59307	2008/12/02 09:53:56.279	00:00:12	log	* Control Syste...	runAp...		INFO			2008-12-02 ...	ky9	ky9m...	CSS	org.c...	Appli...
59306	2008/12/02 09:53:03.780	00:00:52	alarm	STATE	FE_MP...	LOLO...	MAJOR	OK	2		ky9	ky9m...	Ala...		
59305	2008/12/02 09:53:03.768	00:00:00	alarm	STATE	FE_MP...	LOLO...	MAJOR	OK			5hz	pc734...	Ala...		
59304	2008/12/02 09:53:03.576	00:00:00	alarm	STATE	FE_MP...	LOLO...	MAJOR	MAJOR	2		ky9	ky9m...	Ala...		
59303	2008/12/02 09:53:03.566	00:00:00	alarm	STATE	FE_MP...	LOLO...	MAJOR	MAJOR			5hz	pc734...	Ala...		
59302	2008/12/02 09:52:40.071	00:00:23	log	* Control Syste...	runAp...		INFO			2008-12-02 ...	ky9	ky9m...	CSS	org.c...	Appli...
59301	2008/12/02 09:52:34.387	00:00:05	alarm	STATE	CF_ST:...	LOLO...	MAJOR	OK			5hz	pc734...	Ala...		
59300	2008/12/02 09:52:34.376	00:00:00	alarm	STATE	CF_ST:...	LOLO...	MAJOR	OK	3,3...		ky9	ky9m...	Ala...		
59299	2008/12/02 09:52:32.384	00:00:01	alarm	STATE	CF_ST:...	LOLO...	MAJOR	MINOR	3,3...		ky9	ky9m...	Ala...		
59298	2008/12/02 09:52:32.370	00:00:00	alarm	STATE	CF_ST:...	LOLO...	MAJOR	MINOR			5hz	pc734...	Ala...		
59297	2008/12/02 09:52:13.385	00:00:18	log	* Control Syste...	runAp...		INFO			2008-12-02 ...	ky9	ky9m...	CSS	org.c...	Appli...
59296	2008/12/02 09:51:58.461	00:00:14	TALK	MAJOR alarm: ...							5hz	pc734...	Ala...		
59295	2008/12/02 09:51:58.441	00:00:00	alarm	STATE	CF_ST:...	LOLO...	MAJOR	MAJOR			5hz	pc734...	Ala...		
59294	2008/12/02 09:51:58.370	00:00:00	alarm	STATE	CF_ST:...	LOLO...	MAJOR	MAJOR	3,3...		ky9	ky9m...	Ala...		
59293	2008/12/02 09:51:55.391	00:00:02	alarm	STATE	CF_ST:...	LOW_...	MINOR	MINOR	4,8...		ky9	ky9m...	Ala...		
59292	2008/12/02 09:51:55.380	00:00:00	alarm	STATE	CF_ST:...	LOW_...	MINOR	MINOR			5hz	pc734...	Ala...		
59291	2008/12/02 09:51:55.371	00:00:00	TALK	MINOR alarm: ...							5hz	pc734...	Ala...		
59290	2008/12/02 09:51:41.298	00:00:14	alarm	STATE	ICS_Ti...	STAT...	MINOR	OK	Shi...		ky9	ky9m...	Ala...		
59289	2008/12/02 09:51:41.284	00:00:00	alarm	STATE	ICS_Ti...	STAT...	MINOR	OK			5hz	pc734...	Ala...		
59288	2008/12/02 09:51:40.175	00:00:01	alarm	STATE	FE_MP...	LOLO...	MAJOR	OK	2		ky9	ky9m...	Ala...		
59287	2008/12/02 09:51:40.164	00:00:00	alarm	STATE	FE_MP...	LOLO...	MAJOR	OK			5hz	pc734...	Ala...		
59286	2008/12/02 09:51:26.373	00:00:13	alarm	STATE	FE_MP...	LOLO...	MAJOR	MAJOR			5hz	pc734...	Ala...		
59285	2008/12/02 09:51:26.364	00:00:00	alarm	STATE	FE_MP...	LOLO...	MAJOR	MAJOR	2		ky9	ky9m...	Ala...		

By default, the tool displays all messages that were written in the last hour. You can adjust the time range via the “Times” button or by directly entering a start and end time, using for example a start of `-1 days` as shown in the above figure.

The “Filter” button allows you to restrict the message list to for example only log messages, i.e. messages with `TYPE` equal to `log`, and you can further limit the result to messages originating from a specific `HOST`.

The context menu of the message table allows displaying the detail of a message, i.e. it can show all properties of a message. From the context menu you can also export all messages from the table into a file.

Note that there can be many messages, each having many properties. Displaying too many messages would exhaust the available computer memory. By default, the tool restricts itself to only fetching 100000 properties. With the average log message containing 9 properties, this equals roughly 11000 messages. You can adjust this limit in the preference settings of the message history browser.

Chapter 30. Archive Tools - org.csstudio.archive.engine and related

The Archive Engine uses plugins to allow customization. It was developed based on the idea of using a relational database (MySQL, Oracle, PostgreSQL, ...) for storing both the sample engine configuration and the archived data. By replacing for example the plugin that implements the “write” support, one can store samples in a modified RDB table format or even a totally different storage medium. See Chapter 10, *Archive System* for setup and usage of the archive tools.

30.1. org.csstudio.archive.engine

This plugin contains the archive sample engine code. This includes the code that connects to PVs, samples them by various means, as well as a web server to provide access to status information. To obtain the sample engine configuration and to write samples to the actual storage, it uses interfaces from plugins described in the following sections.

The engine plugin also includes an `ArchiveEngine.product` file that is used to build the executable. This product file selects specific implementing plugins, for example to create an RDB-based archive engine. Sites that plan to build the archive engine for a different configuration or sample storage implementation will need to create a different product file.

30.2. org.csstudio.archive.config

This plugin defines the API and extension point for reading an archive engine configuration. It provides access to the list of groups, channels, and the sample configuration of each channel. The archive engine uses this API to determine which channels it should archive and how.

30.3. org.csstudio.archive.config.rdb

This plugin implements the configuration API for an RDB, meaning it reads an archive configuration from MySQL, Oracle or PostgreSQL. It also provides `ArchiveConfigTool.product`, a command-line application that can be used to export a sample configuration into an XML file and (re-)import an archive engine configuration from such an XML file.

By replacing this plugin with a site-specific variant, one could for example store the archive engine configuration in a different table format or even an entirely different system like LDAP.

30.4. org.csstudio.archive.writer

This plugin defines the API and extension point for writing archive samples to a data store. The archive engine uses this API to write samples to the archive.

30.5. org.csstudio.archive.writer.rdb

This plugin implements the sample writer API for an RDB, meaning it allows the archive engine to write samples to MySQL, Oracle or PostgreSQL.

By replacing this plugin with a site-specific variant, one could for example store the archive engine configuration in a different table format or even an entirely different storage system.

30.6. org.csstudio.archive.reader

This plugin defines the API and extension point for reading archived data. It is for example used by the Data Browser to obtain historic data for a channel.

30.7. org.csstudio.archive.reader.rdb

This plugin implements the archive reader API for the RDB table structure used by the RDB-based archive engine. When using an archive engine that writes to a different RDB table layout or even an entirely different storage system, you will have to implement the archive reader API for that system.

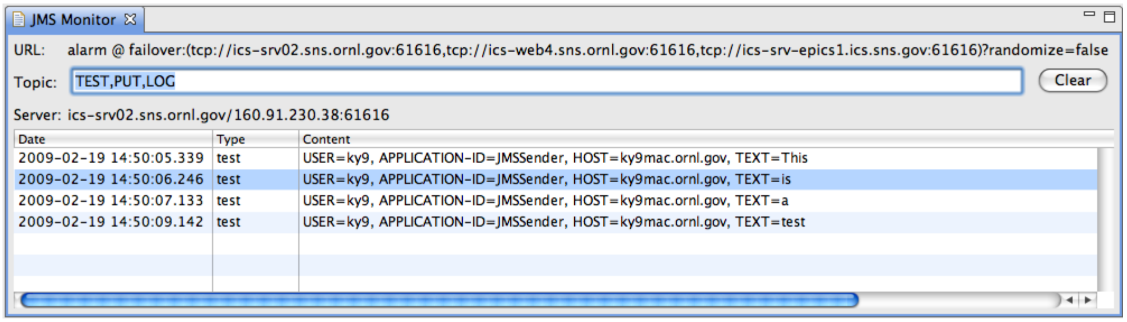
30.8. org.csstudio.archive.rdb

This plugin defines the RDB table structure that is used by the `org.csstudio.archive.config.rdb`, `org.csstudio.archive.writer.rdb` and `org.csstudio.archivereader.rdb` plugins. It also defines the fundamental RDB connection preferences for those plugins, i.e. the RDB URL, user name and password.

Chapter 31. JMS Monitor - org.csstudio.debugging.jmsmonitor

The JMS Monitor is a utility for monitoring raw JMS messages. It is a debug tool for developers.

Figure 31.1. JMS Message Monitor



URL: alarm @ failover:(tcp://ics-srv02.sns.ornl.gov:61616,tcp://ics-web4.sns.ornl.gov:61616,tcp://ics-srv-epics1.ics.sns.gov:61616)?randomize=false

Topic: TEST,PUT,LOG

Server: ics-srv02.sns.ornl.gov/160.91.230.38:61616

Date	Type	Content
2009-02-19 14:50:05.339	test	USER=ky9, APPLICATION-ID=JMSSender, HOST=ky9mac.ornl.gov, TEXT=This
2009-02-19 14:50:06.246	test	USER=ky9, APPLICATION-ID=JMSSender, HOST=ky9mac.ornl.gov, TEXT=is
2009-02-19 14:50:07.133	test	USER=ky9, APPLICATION-ID=JMSSender, HOST=ky9mac.ornl.gov, TEXT=a
2009-02-19 14:50:09.142	test	USER=ky9, APPLICATION-ID=JMSSender, HOST=ky9mac.ornl.gov, TEXT=test

Under its preferences, configure the appropriate JMS connection URL, see Chapter 11, *Java Message Server*.

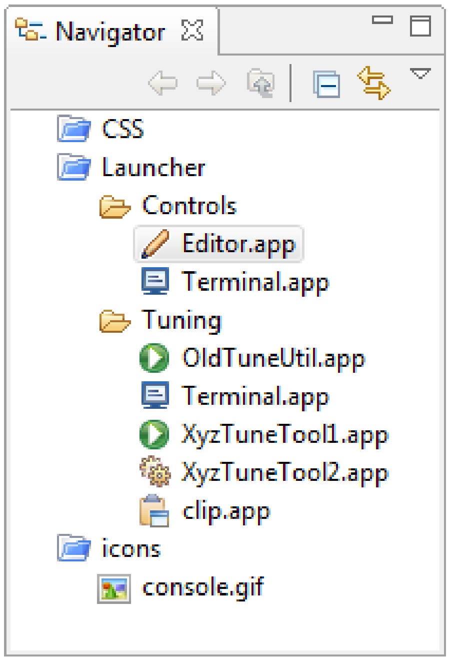
After entering a JMS Topic like `LOG`, `ALARM`, `TALK` etc., it will display received messages. It is possible to enter a list of comma-separated topics.

For more, refer to its online help.

Chapter 32. Application Launcher - org.csstudio.navigator.applaunch

This plugin supports Application Launch configuration files which can then be started from the Eclipse Navigator.

Figure 32.1. Application Launch files



Most accelerator sites have some type of application launcher that lists the many specialized tools that operators or experimenters can start. Users of several subsystem groups can create their own section of the launch configuration. Some tools are listed more than once as shown for the Terminal tool in Figure 32.1, “Application Launch files” because several groups would like to list it in “their” section of the launcher.

This plugin allows using the Eclipse Navigator view as such a launcher. The navigator already displays all workspace files, and can for example open the operator display editor or runtime for display files. It is thus a basic launcher where users can arrange the files in projects and sub-directories and then launch the associated CSS tools. Using linked folders, one can create shared as well as user-specific arrangements.

The use of the Navigator as a launcher complements the CSS menu bar: The menu bar has a comparable limited size and fixed arrangement. It lists the tools built into CSS that many users need. Each CSS tool typically shows only once in the menu bar.

A launcher configuration can be much bigger, listing many external tools. Some entries like the launcher for a terminal window can appear in multiple sub-directories of the launch configuration because each subsystem group of users can create their own section of the launch configuration.

32.1. Basic Usage

End users simply double-click a launch configuration file in the Navigator. Similar to the way double-clicking a display file opens the operator interface tool, opening a launch configuration file will start the associated external application.

Eclipse remembers the last action on a file. If the user had previously edited a launch configuration, i.e. opened it in an editor instead of executing it, Eclipse will remember this and again open the editor on the next double-click. In this case once right-click the launch configuration file and select `Open With`, then `Launch Application` to execute it. From now on the double-click will also default to launching the application.

Users can create new subsections of the launcher by creating new file folders in the Navigator:

- Right-click on the parent folder
- Select `New`, `Other..`, section `General`, type `Folder`
- Enter name of new folder

Since launch configuration files are still files, they can be moved, copied, renamed like any other file in the Navigator.

32.2. Creating a new Application Launcher Configuration File

- Right-click on the parent folder
- Select `New`, `Other..`, section `Launcher`, `Application Launch Configuration`
- Enter a file name, for example `Terminal`. The file extension `.app` will be added automatically, so the resulting launch configuration file would be `Terminal.app`
- On the next page of the Launch Configuration editor, enter the command to launch, for example `xterm` or `cmd.exe`. You may also select an icon.
- Press `Finish`.
- Execute the launch configuration via right-click on the file, selecting `Open With`, then `Application Launcher`. From now on the double-click will also default to launching the application.

32.3. Examples for commands to launch

The command to launch can be anything that is also executable from a shell or command prompt. One example is starting a shell window for entering such commands.

On Windows, this could be `cmd.exe` which is in the `%PATH%`.

On Linux, it could be `xterm` if on the `$PATH`, or better `/usr/bin/X11/xterm` or a similar complete path.

On Mac OS X, one could use `/Applications/Utilities/Terminal.app`. Beware: Both the Launch configuration file names and the Mac OS application directories use an extension `*.app`. In your CSS workspace you can create a file `Terminal.app` that contains the launch configuration for the Mac OS X terminal window application `/Applications/Utilities/Terminal.app`. Running that launcher in CSS will then start the Mac OS X terminal application.

If you want to launch a program that requires additional command-line arguments, you can create a batch file or shell script to invoke the program as desired and then use that batch file or shell script as the launcher command.

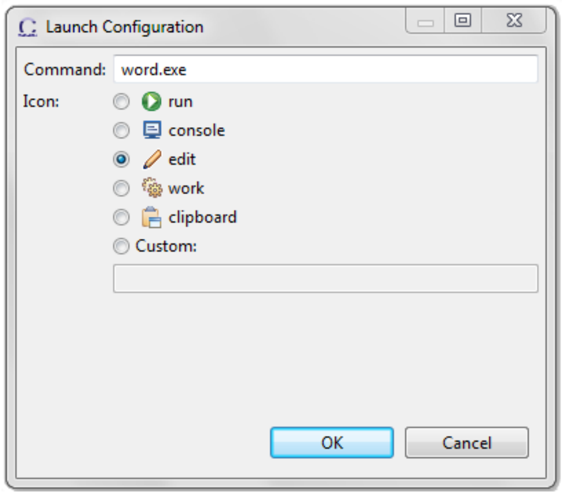
On Windows or Mac OS X the command can also be the path to a file where the operating system knows how to handle that file. For example, if double-clicking on a `*.doc` file will open MS Word on your computer, you can use the full path to such `*.doc` files as launcher commands. When activating such a command, the operating system will open MS Word with the document.

32.4. Editing an Application Launcher Configuration File

To edit an existing launcher configuration file:

- Right-click on the `*.app` file that you want to edit
- Select `Open With`, then `Application Editor`.

Figure 32.2. Configuration Editor



Command: word.exe
Icon: ( ) run ( ) console [✓ edit] ( ) work ( ) clipboard ( ) Custom:

The launcher configuration editor, see Figure 32.2, “Configuration Editor”, allows you to modify the command to execute and to select a different icon.

32.5. Application Launcher Configuration File Details

If you create launcher configuration files by other means, you need to be familiar with its XML file format.

The application launcher configuration file name must end in `.app`. It must be a valid XML file with an `application` root element and a `command`:

```
<application>
  <command>/path/to/the/command</command>
</application>
```

The command should be the path to a command. On MS Windows and Mac OS X, the command may also the path to a file that the operating system can open. For example, it could be the path to a Microsoft

Word document and invoking the command would then open that file in Microsoft Word on a PC or Mac that has MS Word available.

The launcher configuration file may include an optional `icon`:

```
<application>
  <command>/path/to/the/command</command>
  <icon>icon_info</icon>
</application>
```

The icon info can be the following:

- `icon:clipboard`, `icon:console`, `icon:edit`, `icon:run`, `icon:text`, or `icon:work` to select built-in icons. The default is `icon:run`.
- A path to an icon file within the workspace. In an operational setup one might prepare a shared icon folder like `CSS/Share/icons`, and then use an icon path like `CSS/Share/icons/some_icon.gif` as a launcher icon.

Chapter 33. RDB Logging - org.cstudio.sns.jms2rdb

The plugin `org.cstudio.sns.jms2rdb` is a tool that listens to JMS messages from the log, alarm system or other CSS applications and sends them to the RDB.

This tool uses an RDB schema for messages that extends the basic schema described in Section 11.5, “JMS logging to RDB” by adding the following commonly used message properties to the MESSAGE table itself:

- `TYPE`: The message type, for example “log” or “alarm”.
- `SEVERITY`: The severity of the message, for example “ERROR” or “MINOR”.
- `NAME`: A name associated with the message. For “log” messages, this is typically the name of the Java method that logged the information. For “alarm” messages, this is the name of the Process Variable that triggered the alarm.

33.1. Relational Database Setup

The `dbd` sub-directory of the plugin sources describes the RDB schema for MySQL, Oracle and PostgreSQL.

33.2. Building the Tool

Use the `JMS2RDB.product` file to generate the binary.

33.3. Running the Tool

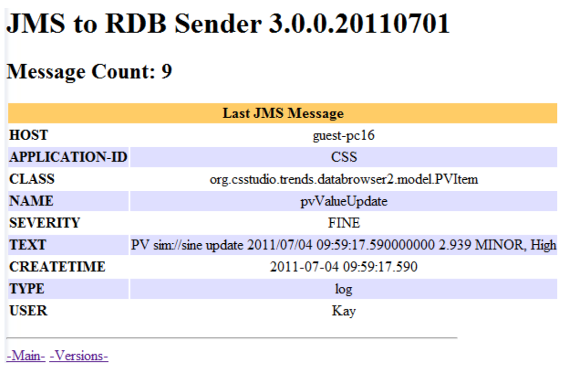
You configure the connection parameters for JMS and your RDB via a plugin customization file as described in Chapter 6, *Hierarchical Preferences*. Refer to the file `plugin_customization.ini` included in the JMS2RDB plugin sources for an example.

Once running, the JMS2RDB tool provides a web browser interface as a basic status monitor under the URL

```
http://localhost:4913/main
```

The port number can be adjusted in the preference settings of the tool. To connect from another computer, replace `localhost` with the name of the host that is executing the tool.

Figure 33.1. JMS-2-RDB Web Interface



JMS to RDB Sender 3.0.0.20110701

Message Count: 9

Last JMS Message	
HOST	guest-pc16
APPLICATION-ID	CSS
CLASS	org.csstudio.trends.databrowser2.model.PVItem
NAME	pvValueUpdate
SEVERITY	FINE
TEXT	PV sim://sine update 2011/07/04 09:59:17.590000000 2.939 MINOR, High
CREATETIME	2011-07-04 09:59:17.590
TYPE	log
USER	Kay

-Main- -Versions-

The main web page will look similar to Figure 33.1, “JMS-2-RDB Web Interface”, displaying the last message that was received and written to the RDB. In case of errors, an addition “Last Error” section will display for example the last database access error.

To stop the tool, access the URL

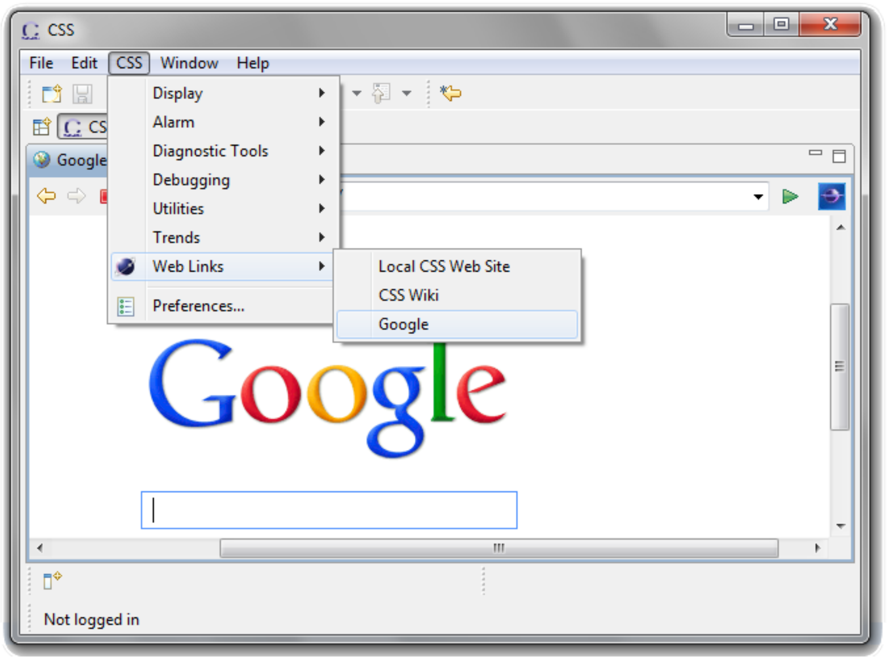
```
http://localhost:4913/stop
```

Chapter 34. Web menu - org.csstudio.ui.menu.web

The plugin “org.csstudio.ui.menu.web” adds a `Web` entry to the `CSS` menu which allows users to access web sites related to the control system.

As a growing number of control system related tools become web-based, this allows easy access to them from within CSS.

Figure 34.1. Web menu



Depending on the operating system and the preference settings under `General`, `Web Browser`, the web pages will open in a web browser within the CSS workbench or in an external web browser.

34.1. Configuration

The web links are configured by the CSS maintainer for each site via preferences of the “org.csstudio.ui.menu.web” plugin.

The example web menu from Figure 34.1, “Web menu” was created with the following configuration placed in the `plugin_customization.ini` of the product:

```
# Selects the web links to show and define their order.
# When left blank, there won't be any web links.
org.csstudio.ui.menu.web/weblinks=local css google

# Define the Label and link for each web link.
# Only those listed in ...weblinks above are actually used!
# Link to the local CSS web site where users can download CSS,
# learn about updates etc:
org.csstudio.ui.menu.web/local=\
Local CSS Web Site|http://www.my-institute.edu/css/

# The main CSS web page on Source Forge
org.csstudio.ui.menu.web/css=\
CSS Wiki|https://sourceforge.net/apps/trac/cs-studio

# Example for other useful links
org.csstudio.ui.menu.web/google=Google|http://www.google.com
```

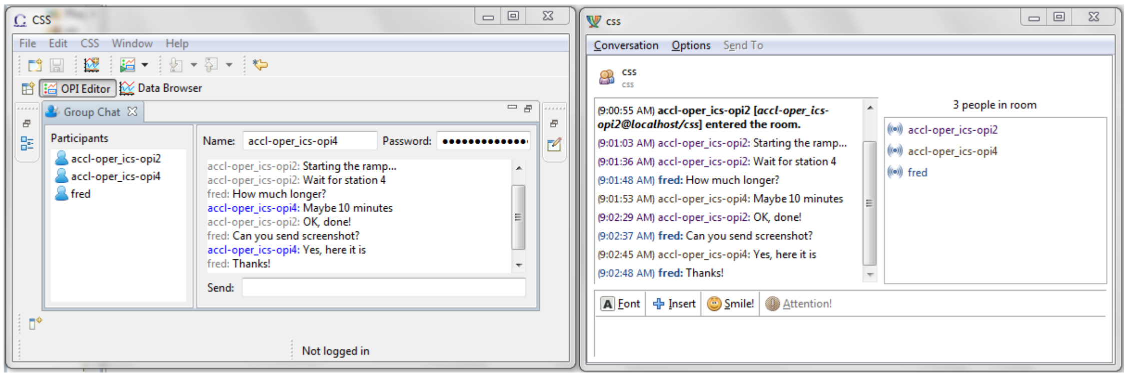
Chapter 35. Chat - org.csstudio.utility.chat

The Chat tool is a basic online group chat client for CSS. The emphasis is on use in a control room context:

- Easily log onto control room chat group.
- Simply exchange screen-shots and files.

Technically, it is an XMPP chat protocol client that can communicate with other clients that support the same protocol (Pidgin, Google chat, Mac OS iChat, ...).

Figure 35.1. Chat Client (left) communicating with “Pidgin” (right)



Participants: accl-oper_ics-opi2, accl-oper_ics-opi4, fred

Name: accl-oper_ics-opi4

```
accl-oper_ics-opi2: Starting the ramp...
accl-oper_ics-opi2: Wait for station 4
fred: How much longer?
accl-oper_ics-opi4: Maybe 10 minutes
accl-oper_ics-opi2: OK, done!
fred: Can you send screenshot?
accl-oper_ics-opi4: Yes, here it is
fred: Thanks!
```

```
(9:00:55 AM) accl-oper_ics-opi2 [accl-oper_ics-opi2@localhost/css] entered the room.
(9:01:03 AM) accl-oper_ics-opi2: Starting the ramp...
(9:01:36 AM) accl-oper_ics-opi2: Wait for station 4
(9:01:48 AM) fred: How much longer?
(9:01:53 AM) accl-oper_ics-opi4: Maybe 10 minutes
(9:02:29 AM) accl-oper_ics-opi2: OK, done!
(9:02:37 AM) fred: Can you send screenshot?
(9:02:45 AM) accl-oper_ics-opi4: Yes, here it is
(9:02:48 AM) fred: Thanks!
```

accl-oper_ics-opi2
accl-oper_ics-opi4
fred

35.1. Basic Usage

Start the chat client from the menu `CSS`, `Utilities`, `Group Chat`.

The suggested user name that is initially displayed in the `Name:` field of the chat tool is the name of the user logged onto the computer suffixed by the network name of the computer. The default password will be a dollar sign followed by the user name, and the tool will attempt to create an account for that user and password on the chat server. The idea is that most users can simply press the “Return” respectively “Enter” key in the user name field to log on with the suggested user name and password. Advanced users may enter a user name and password that was registered with the chat server by other means.

Once connected to the group chat, you can see other participants who are connected to the chat in the `Participants` list. Enter text to send in the `Send:` field, completing lines to send with the “Return” key.

35.2. Sending Files

By right-clicking on one of the entries in the `Participants` list you can access a context menu that allows you to “Contact” another chat participant. A separate chat panel will open for communicating with only that user, outside of the group chat. Note that the other participant will be asked if she accepts the invitation to an individual chat, and may decline.

35.3. Individual Chats

By right-clicking on one of the entries in the `Participants` list you can access a context menu with options “Send File” and “Send Screenshot”. When selecting to send a screen-shot, an image of your current desktop will be sent to the receiver. When sending a file, you will be prompted for the name of the file that you want to send. Note that in either case the receiving participant of the chat will be asked if she accepts the file that you sent, and may decline.

35.4. XMPP Server Setup - Openfire

XMPP is an open technology for real-time communication. More information can be found at http://xmpp.org, including lists of XMPP server implementations and other chat clients compatible with XMPP.

`Openfire`, available from http://www.igniterealtime.org/projects/openfire, is an open source, Java-based XMPP server for Windows, Linux and Mac OS X. For Windows, it includes an executable `bin/openfire.exe` to start and stop the server and to access its web interface via a “Launch Admin” button.

When starting Openfire for the first time, you need to configure the following via its web interface:

- Domain - Set to either `localhost` for initial tests on one computer, or set to the complete and correct host name.
- Database - Selecting the “embedded” database seems to work fine.
- Administrator account - Use for example “admin@localhost” and a password of your choice.

From now on, to log into the admin web interface of Openfire, you have to use just “admin” and the password that you configured, **not** “admin@localhost”!

In the Openfire online admin interface you should check the setting under “Server”, “Server Settings”, “Registration & Login”. By default, users should be allowed to automatically create new accounts. The CSS chat client will use this option to attempt automatic account creation with the suggested user name and password as described in Section 35.1, “Basic Usage”. If you choose to disable automated account creation, you need to create user accounts in the Openfire admin interface.

The CSS chat client expects to connect to a chat room called “css”. From the Openfire online admin interface, select the “Group Chat” tab. Press “Create New Room”:

- Enter `css` as the room ID.
- You will also need to enter a room name, description and topic, which you can all set to “CSS”.
- Under `Show Real JIDs of Occupants to:`, select “Anyone”. This will allow the chat clients to directly contact chat group participants for individual chats and file exchange.
- Before you save the room settings, take note of the full name of the group chat. It should use the chat room ID followed by `@conference` and the host name of the server, similar to

  ```
  css@conference.localhost
  ```

To reset the Openfire configuration, for example after forgetting the admin password, stop openfire. In the file `conf/openfire.xml`, locate the `setup` entry. Set it to `false`, start Openfire again and when you now access the admin web interface, you can configure a new admin password.

35.5. Chat Client Settings

The CSS chat clients has the following preference settings:

- chat_server - Host name that runs the chat server, for example “localhost”.
- group - Name of the chat group to join, for example “css@conference.localhost”.

35.6. Example using Pidgin

Pidgin, available from http://pidgin.im, is an open-source chat client for various operating systems that can be used to test an XMPP server setup.

To configure Pidgin for a connection with Openfire, use the Pidgin menu bar to create an account with the following settings:

- Procotol - XMPP
- Username, password - Either enter a name and password that has already been added to Openfire, or enter a new name and password with the option to “Create this new account on server”.
- Domain - Enter name of XMPP server host, for example “localhost”.

In the “Advanced” account setup panel, check that the XMPP connect port is set to 5222 and the file transfer proxies is set to the host name of your XMPP server port 7777, for example `localhost:7777`.

Appendix A. Docbook

This document is written in DocBook. The Docbook XML format offers a single-source platform for generating HTML, PDF and other outputs. The generated HTML is very minimal, much less verbose than for example the HTML generated by word processors.

There are many online resources for the DocBook syntax, for example http://www.docbook.org/tdg5/en/html/docbook.html. While these nicely describe the syntax, I had a hard time finding a concise description of a tool set for translating DocBook into HTML or PDF. The following seems to work on Linux and OS X.

A.1. Example Document

This is an example document:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://docbook.org/ns/docbook" version="5.0">
 <title>Very simple book</title>
 <chapter>
   <title>Hello</title>
   <para>Hello world!</para>
 </chapter>
</book>
```

Most real-world documents would actually be split across files, for example a `example.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://docbook.org/ns/docbook" version="5.0"
      xmlns:xi="http://www.w3.org/2001/XInclude">
 <title>Very simple book</title>
 <xi:include href="chapter.xml" />
</book>
```

.. which includes a `chapter.xml` using XInclude:

```
<?xml version="1.0" encoding="UTF-8"?>
<chapter xmlns="http://docbook.org/ns/docbook" version="5.0">
  <title>Hello</title>
  <para>Hello world!</para>
</chapter>
```

A.2. Style Sheets, Processor

XSL translations are used to convert the DocBook XML into other formats. The necessary set of style sheets, for example the version `docbook-xsl-1.76.1`, can be downloaded from http://sourceforge.net/projects/docbook/files.

In addition to the style sheets, a processing tool that can apply XSL translations to XML documents is required. Linux and Mac OS X already include such a tool, namely `xsltproc`. For Windows, you need to find such a tool.

A.3. Generate HTML

HTML is generated by simply applying the appropriate translation:

```
xsltproc /path/to/docbook-xsl-1.76.1/html/docbook.xsl \
         example.xml  >example.html
```

A.4. Generate PDF

Fundamentally, PDF is created by first translating the DocBook XML into an intermediate format like LaTeX, then using LaTeX to generate PDF. A very convenient intermediate format is XSL-FO because the free, open source Apache Java FOP tool can perform the transformation to FO and render the result as PDF.

After downloading and installing Apache FOP, use a command like this:

```
fop -xsl /path/to/docbook-xsl-1.76.1/html/docbook.xsl \
    -xml example.xml  -pdf example.pdf
```

A.5. Options

Both `xsltproc` and `fop` support processing options that influence the generated output:

html.stylesheet	Set to the name of a Cascading Style Sheet. Only applies to HTML output.
generate.toc	Set to `0` to disable table of contents.
generate.index	Set to `0` to disable index.
chapter.autolabel	Set to `0` to disable numbering of chapters.

For `xsltproc`, they are passed as `xsltproc --stringparam parm value ...` For `fop`, use `fop -param parm value ...`

Index