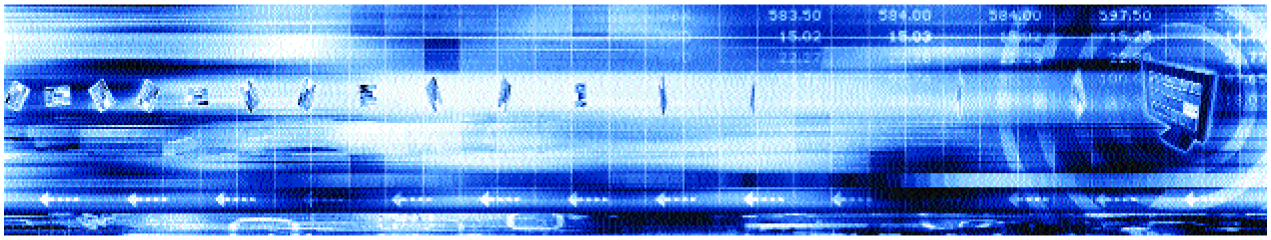


PriceMaster Admin Guide

September 2006

CAPLIN

Table of Contents

Section I

Preface

What this document contains

This document gives an overview of Caplin's PriceMaster product and describes how to install, configure and manage it.

Who should read this document

This document is intended for System Administrators and Operators who need to install, configure and manage PriceMaster.

Typographical conventions

This document uses the following typographical conventions to identify particular elements within the text.

Type	***Uses***
Arial Bold	Function names and methods
Arial Italic	Parameter names and other variables.
Times Italic	File names, folders and directories
`Courier`	Program output and code examples.
❖	Information bullet points
▪	Instruction

Important Notes are enclosed within parallel red lines as below, pay particular attention to these points to ensure proper configuration and operation of the solution.

Note: This is a particularly important point.

Useful Tips are enclosed within parallel green lines as below, use these points to find out where to get more help on a topic.

Tip: This is a useful piece of information.

Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

Please email your thoughts to documentation@caplin.com.

Section II

Product Overview

2.1 What is PriceMaster?

PriceMaster is a comprehensive middleware engine for controlling, converting, permissioning and distributing real-time market data. It is capable of collecting and converting information from Caplin's DataSource market data platform, a variety of industry-standard data feeds and common trading system middleware solutions for onwards publication to traditional quote vendors, trading hubs, data aggregators and Caplin's Liberator (for onward distribution over the internet).

PriceMaster provides complete control over the conversion, permissioning, format, presentation and distribution of data, eliminating fragmentation and complexity. It combines:

- Instant data acquisition from almost all conventional sources
- Any-to-any format conversion
- Page construction
- Comprehensive symbology and data translation on a per-output basis
- Highly secure permissioning
- Publishing of data over internal trading systems and other networks
- Connection to Caplin Liberator for publishing live data on the web

PriceMaster provides a secure path to any chosen information destination, automatically recoding data as required. Permissioning, symbology translation and data translation is available down to the level of individual instruments, with a single point of control.

PriceMaster can integrate with any DataSource-enabled middleware system. For example, when integrating with Reuters Triarch / RMDS, PriceMaster can function either as a Source Sink Library (“ SSL”) source or a sink (or both) as required. It can therefore receive both SSL source data and SSL inserts with equal ease. Contribution then takes place exactly as it would with a standard Triarch DCS.

2.2 Features of PriceMaster

Data Input

PriceMaster is able to receive data from any application using a DataSource Adaptor. Adaptors exist for most platforms including Triarch, RMDS, TIB, RV, ILX, Hyperfeed, Comstock and Dow Jones News. Custom input adapters can be created using Caplin’s DataSource SDKs which are available for C/C++ and Java.

Data Output

Output from PriceMaster is accepted by Reuters, Bloomberg, Bridge, MoneyLine, CQG, TradeWeb or direct to any other DataSource enabled adaptor including Caplin Liberator for publishing over the internet. Custom output adapters can also be created using Caplin’s DataSource SDKs which are available for C/C++ and Java.

Data Permission Control

PriceMaster incorporates an XML based permissioning module which allows permissions to be applied to individual objects for each output. It also allows for automatic authorization to be applied to objects using pattern matching of object names for each vendor output.

Symbology and data mapping

PriceMaster includes configurable mapping of object name and field name/number on a per object and per output basis. In addition, formatting rules can be applied to these objects on a per output basis. This allows different objects to be mapped to the same names for different vendors, or perhaps different business rules to be applied to the same objects for different vendors.

Page Construction

PriceMaster supports end-to-end traditional page-based contribution. In addition, pages can be constructed by PriceMaster from input records, sections of other pages or text. Complex column-based layouts can be created and formatting and alignment rules applied to the data within the page. These are defined using an XML format which can be edited either manually or more typically using the GUI console.

Extensible formatting module

PriceMaster provides an extensible formatting module for use by all parts of the system. One or more formatting algorithms can be applied to record fields or page data areas. The standard supported formatters include decimal places, whitespace stripping, fraction conversion, date formatting, alignment, bond notation. Additional formatters can be created using the Transformer SDKs which are available for C/C++ and Java.

Easy to use control and configuration GUI

PriceMaster integrates with Caplin's cross-platform Enterprise Management Console and provides a set of custom views which provide monitoring and control of the system. The views include status of input and output feeds, permission and data mapping configuration, page construction and inspection of the data and logs.

Extensibility

PriceMaster is easily extensible using Transformer modules to add new server functionality and the Monitoring and Management SDK to create new GUI views. The Transformer SDKs are available for C/C++ and Java, Transformer business logic can also be created using the new scriptable pipeline module.

Monitoring and Management

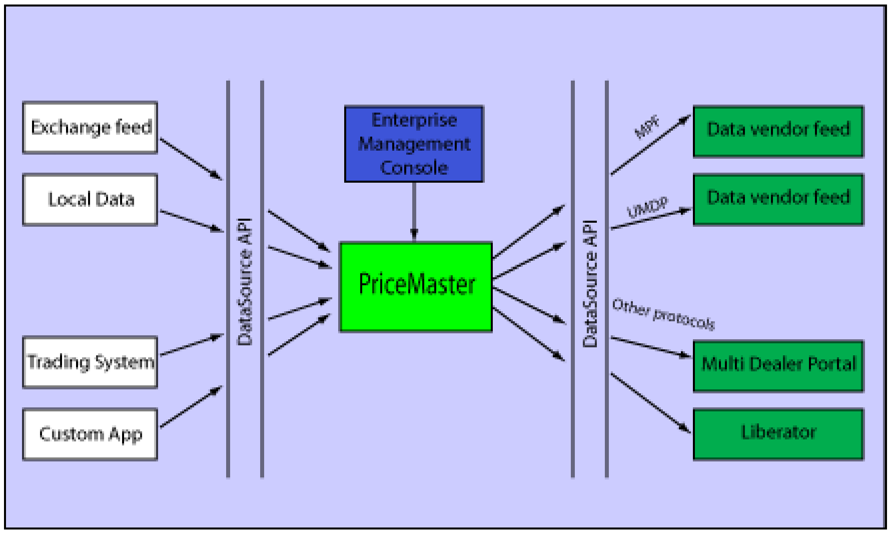
PriceMaster has a JMX interface which allows you to connect to your existing monitoring applications or create custom monitoring and control views on PriceMaster.

2.3 PriceMaster and the Caplin Platform

PriceMaster integrates fully with other products and components of the Caplin platform.

- PriceMaster is built on Caplin's Transformer product and benefits from the reliability, resilience, flexibility and modularity of the platform.
- Data is received and sent by PriceMaster using Caplin's DataSource protocol, meaning it can comunicate with any existing DataSource adapter or custom developed DataSource for input and output.
- PriceMaster can be monitored and managed using Caplin's Enterprise Management Console or customer-built JMX or socket monitoring application.
- PriceMaster contains a suite of custom modules to provide complete control over the permissioning, formatting and output symbology of the data it processes.

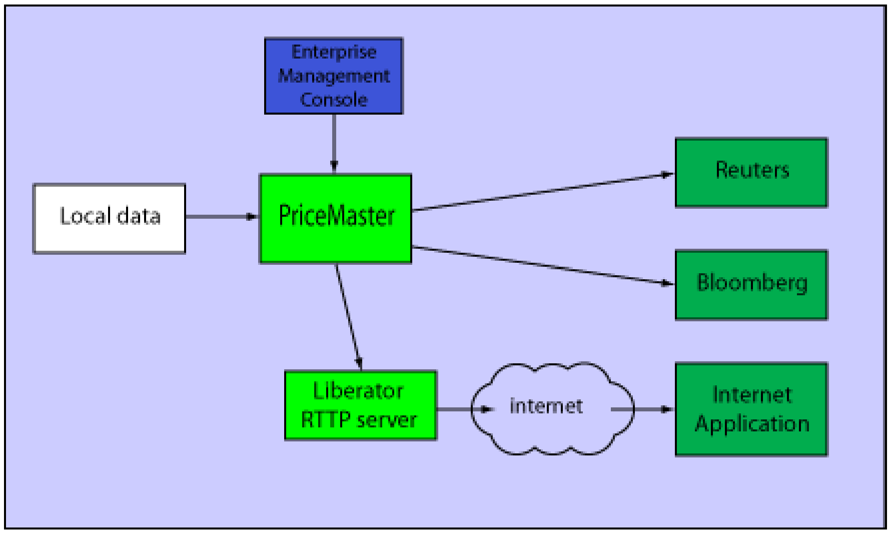
The diagram below illustrates the flow of data through PriceMaster 4.



Flow of data through PriceMaster 4

DataSource adaptors interface with exchange feeds, market data platforms such as Triarch and RMDS, trading systems and custom applications. These pass the data to the PriceMaster using the DataSource protocol. PriceMaster performs operations such as page construction, permissioning and formatting then passes the data using a DataSource enabled output handler to a vendor feed, multi dealer portal, Caplin DataSource or custom application. The Enterprise Management Console is used to control, configure and monitor PriceMaster.

In addition to providing data to traditional data provider feeds or trading portals, PriceMaster can connect to Caplin's Liberator server which can then feed the data to any RTTP enabled front-end. The diagram below shows a typical PriceMaster configuration with 2 vendor output connections and a Liberator distributing data directly over the internet to a user's application.



Flow of data through PriceMaster and Liberator to an internet application

Section III

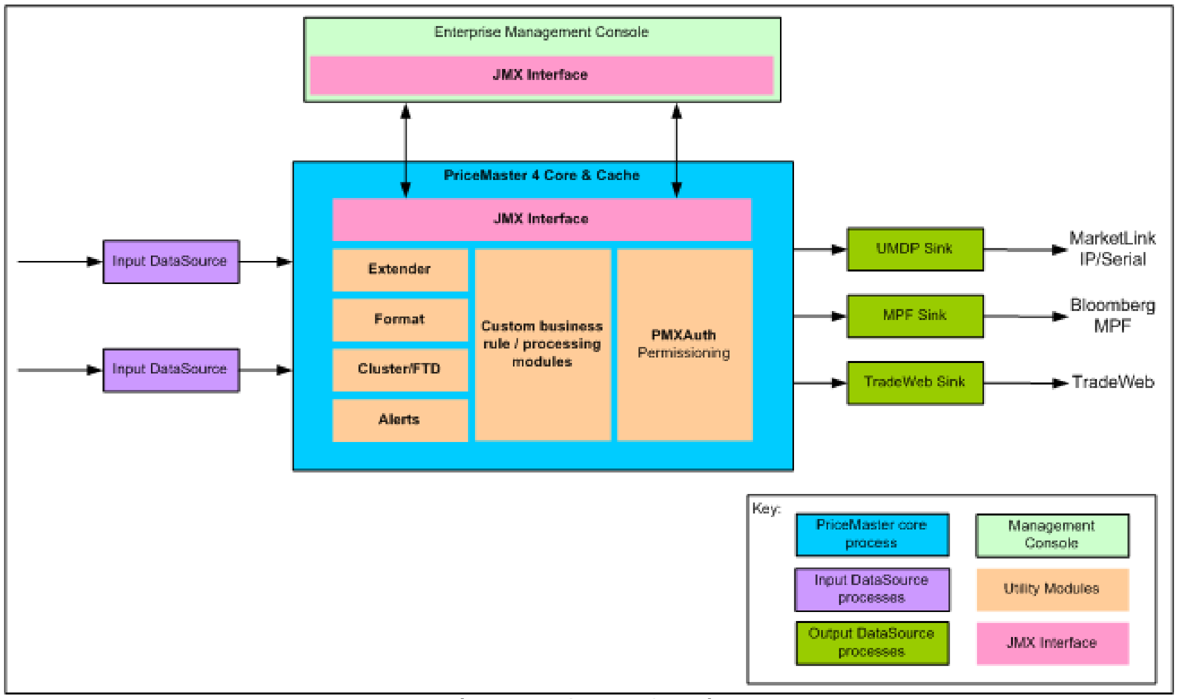
System Overview

3.1 PriceMaster architecture

PriceMaster consists of a number of server processes and a console GUI.

The following diagram shows the architecture of components and modules that make up the PriceMaster product. Each box that is edged with a solid line is an individual process, boxes without solid lines represent loaded modules or libraries used by the components. The arrows show the communication between the processes. The input and output processes communicate using Caplin's DataSource protocol and the Enterprise Management Console communicates using JMX remoting API (JSR160).

Note - this diagram shows an installation with 2 inputs and 3 outputs, though up to 63 inputs / outputs may be configured.



PriceMaster System Overview

3.2 PriceMaster server processes

The PriceMaster server consists of a number of different processes communicating using Caplin's DataSource protocol.

There is a server process for the main PriceMaster core and one for each output handler instance. The PriceMaster and its processes are started and stopped using a single script: etc/pricemaster.

PriceMaster Core

The PriceMaster core process is built on Caplin's Transformer product. This is an modular and extensible business rules and data processing engine. The PriceMaster functionality is provided by a suite of modules to provide caching, formatting, page construction, permissioning, clustering and any custom logic required.

Formatter Module (format)

The formatter module provides standard formatting functions to all modules of the PriceMaster core process. The following formatting functionality is supported out of the box:

- Alignment (left, right, centre)
- Significant figures
- Decimal place rounding
- Bond formatting
- Fraction to decimal conversion
- White space removal
- Timestamps

Additional formatting options can be created by custom modules and registered with the format module which then permits any module within the system to reference the new format style.

Fault Tolerant Module (cluster)

This module implements the basic fault tolerant behaviour that is required to synchronise PriceMaster instances. This module is used to synchronise data and configuration between PriceMaster instances.

PriceMaster Permissioning Module (pmxauth)

The permissioning module provides authorisation services for the vendor outputs. The module permits the mapping of updates from local format to a format preferred by the vendor.
The mapping may take the form of a simple name mapping, or more complex mapping of field names to strings or numbers (for example ASK to ASK_1 or ASK to 4723).
Permissions and mapping information is described in an XML file for each vendor. Revision history is available to provide an audit trail of permission changes.

Page Construction Module (pagecons)

The page construction module permits the construction of page based data from incoming data. Page

definitions are stored in an XML format that is manipulated using the Enterprise Management Console. Revision history is available to provide an audit trail of page template changes.
Pages can be constructed out of the following elements:

- Static text
- Record field data
- Areas of other pages

Record field data can be formatted using any of the formatting rules provided by the format module prior to insertion into the page.

Output DataSources (Sinks)

UMDP Sink

This output handler provides contributions to vendors that accept UMDP (MarketLink) protocol data over either TCP/IP or serial connections.

MPF Sink

This output handler provides Bloomberg MPF contributions, exclusively to Bloomberg.

TradeWeb Sink

This output handler provides vendor contribution to TradeWeb.

3.3 Enterprise Management Console for PriceMaster

The Enterprise Management Console is Caplin's cross-platform monitoring and management GUI.

It is a Java application that works on any platform with Java 1.5.0 or newer. See the section on Starting the Enterprise Management console for more details.

PriceMaster 4 is supplied with a set of custom views for the Enterprise Management Console which allow the control and configuration of PriceMaster using a simple graphical interface.

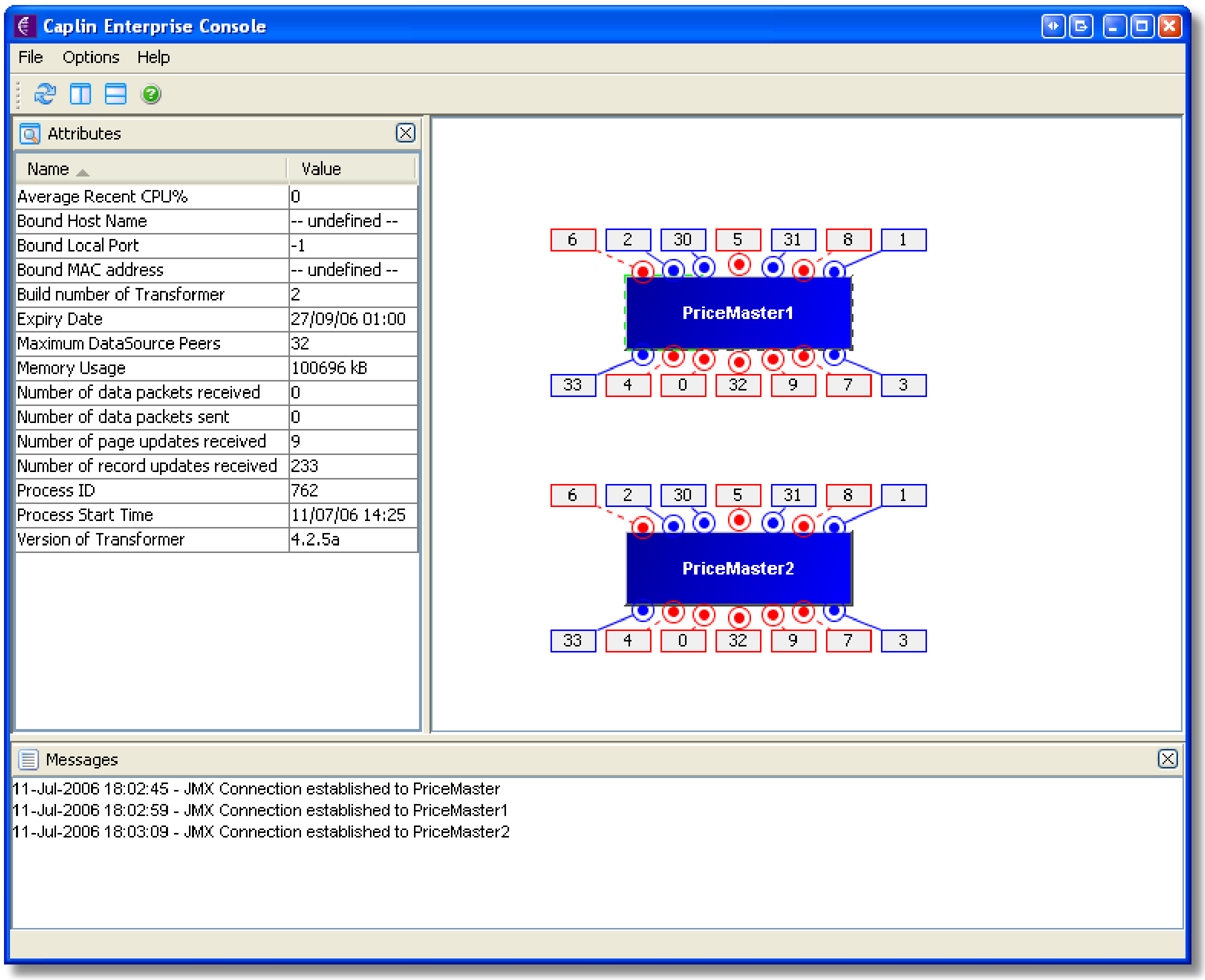
The console can be used to monitor input feeds, permission map and format data for output feeds, create pages of data and view or resend the values of any objects currently in the PriceMaster. The console has a comprehensive online help guide, please refer to it for full details on how to use each of these views.

The Management Console for PriceMaster 4 includes the following functional areas:

- Enterprise Console
- PriceMaster Overview
- PriceMaster Permissions Editor
- PriceMaster Page Constructor
- Alerts Configuration Editor
- PriceMaster Administration
- Explorer

Enterprise Console

The Enterprise Console is the initial application window seen when starting the Enterprise Management Console. This view allows all Caplin JMX-enabled server products to be monitored and managed through a single interface. The following screenshot shows the Enterprise Console configured to connect to two PriceMaster instances, and quickly relays the status of those PriceMaster applications. By double-clicking on one of the PriceMaster instances the PriceMaster Console window will open, containing the PriceMaster custom views detailed below.



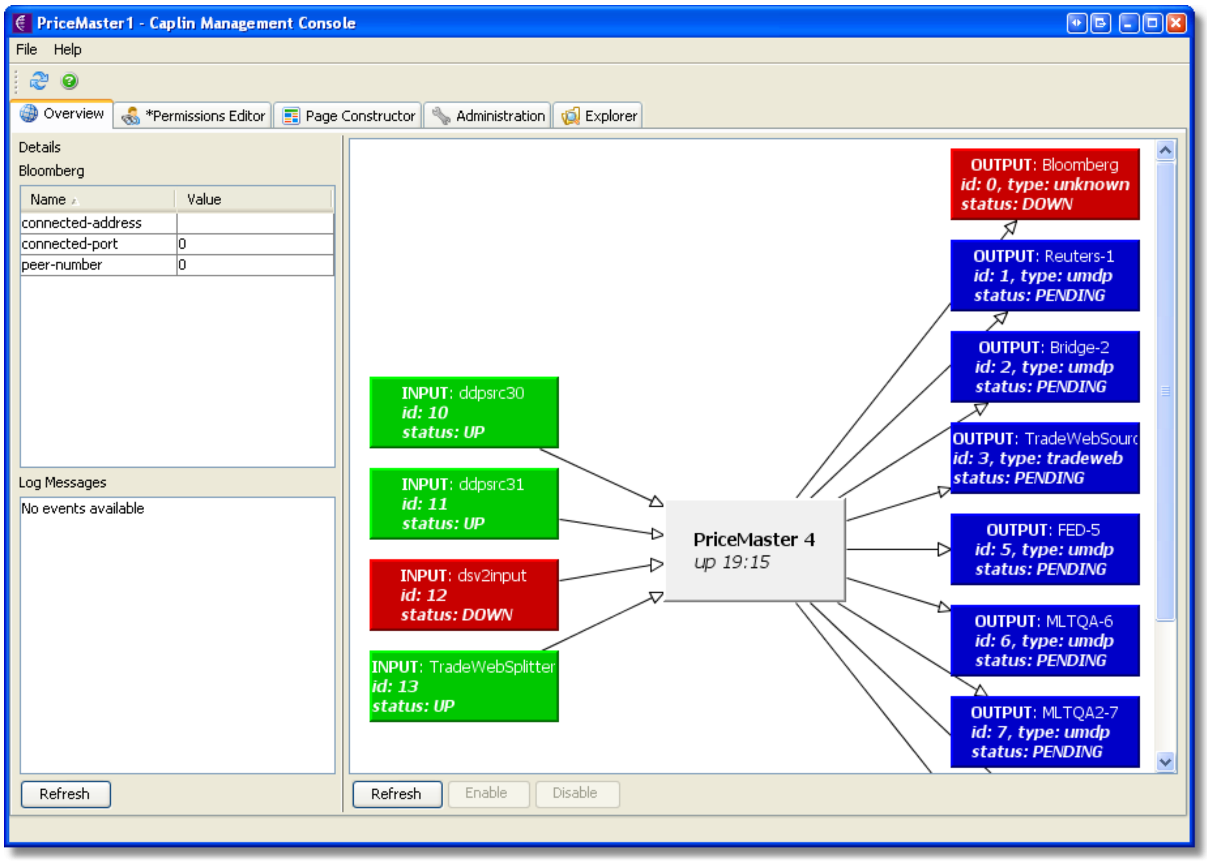
Name	Value
Average Recent CPU%	0
Bound Host Name	-- undefined --
Bound Local Port	-1
Bound MAC address	-- undefined --
Build number of Transformer	2
Expiry Date	27/09/06 01:00
Maximum DataSource Peers	32
Memory Usage	100696 kB
Number of data packets received	0
Number of data packets sent	0
Number of page updates received	9
Number of record updates received	233
Process ID	762
Process Start Time	11/07/06 14:25
Version of Transformer	4.2.5a

```
11-Jul-2006 18:02:45 - JMX Connection established to PriceMaster
11-Jul-2006 18:02:59 - JMX Connection established to PriceMaster1
11-Jul-2006 18:03:09 - JMX Connection established to PriceMaster2
```

The Enterprise Console

PriceMaster Overview

The PriceMaster Overview diagrammatically shows the input and output feeds connected to the PriceMaster. Boxes to the left of PriceMaster on the diagram represent input feeds, boxes to the right represent output feeds. Clicking on the boxes representing input and output feeds in the diagram displays data about the connection in the top left Details panel and that process's log messages in the bottom-left Log Messages panel. Clicking on PriceMaster itself shows process, license and connection information about the PriceMaster.


PriceMaster1 - Caplin Management Console
File
Help
Overview
*Permissions Editor
Page Constructor
Administration
Explorer
Refresh
Enable
Disable
INPUT: ddpsrc30 id: 10 status: UP
INPUT: ddpsrc31 id: 11 status: UP
INPUT: dsv2input id: 12 status: DOWN
INPUT: TradeWebSplitter id: 13 status: UP
PriceMaster 4 up 19:15
OUTPUT: Bloomberg id: 0, type: unknown status: DOWN
OUTPUT: Reuters-1 id: 1, type: umdp status: PENDING
OUTPUT: Bridge-2 id: 2, type: umdp status: PENDING
OUTPUT: TradeWebSourc id: 3, type: tradeweb status: PENDING
OUTPUT: FED-5 id: 5, type: umdp status: PENDING
OUTPUT: MLTQA-6 id: 6, type: umdp status: PENDING
OUTPUT: MLTQA2-7 id: 7, type: umdp status: PENDING

Details

Bloomberg

Name	Value
connected-address	
connected-port	0
peer-number	0

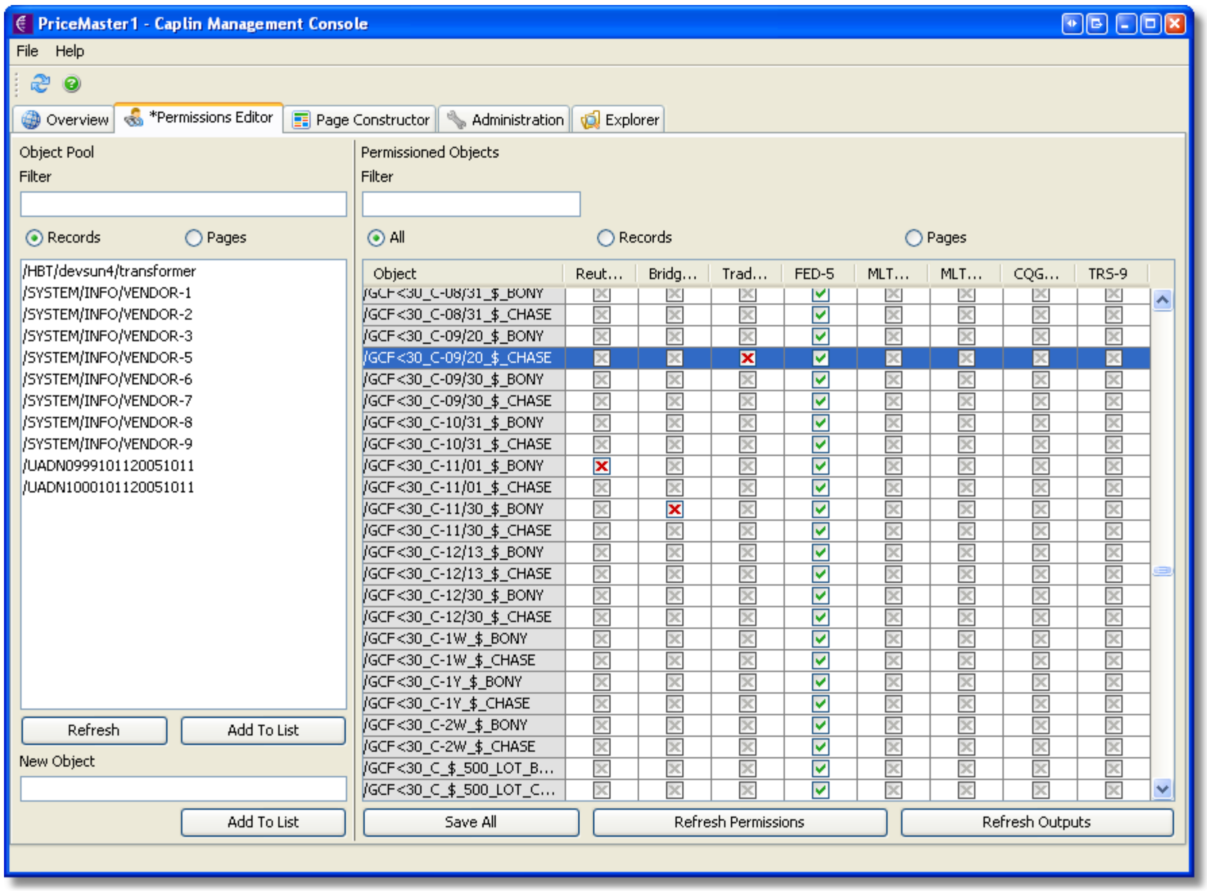
Log Messages

No events available

The PriceMaster Overview

PriceMaster Permissions Editor

The Permissions Editor enables you to set up a sophisticated permissioning system defining which outputs are able to receive which objects. As a user you are able to search for objects being received by PriceMaster and then add them to your permissioning list, setting the permissions as appropriate against each of the outputs to which you are connected. The Permissions Editor also provides per-output mapping of object names and field names / numbers through context-sensitive menus on the main permissions table.



Object Pool (Records [✓]):

/HBT/devsun4/transformer
/SYSTEM/INFO/VENDOR-1
/SYSTEM/INFO/VENDOR-2
/SYSTEM/INFO/VENDOR-3
/SYSTEM/INFO/VENDOR-5
/SYSTEM/INFO/VENDOR-6
/SYSTEM/INFO/VENDOR-7
/SYSTEM/INFO/VENDOR-8
/SYSTEM/INFO/VENDOR-9
/UADN0999101120051011
/UADN1000101120051011

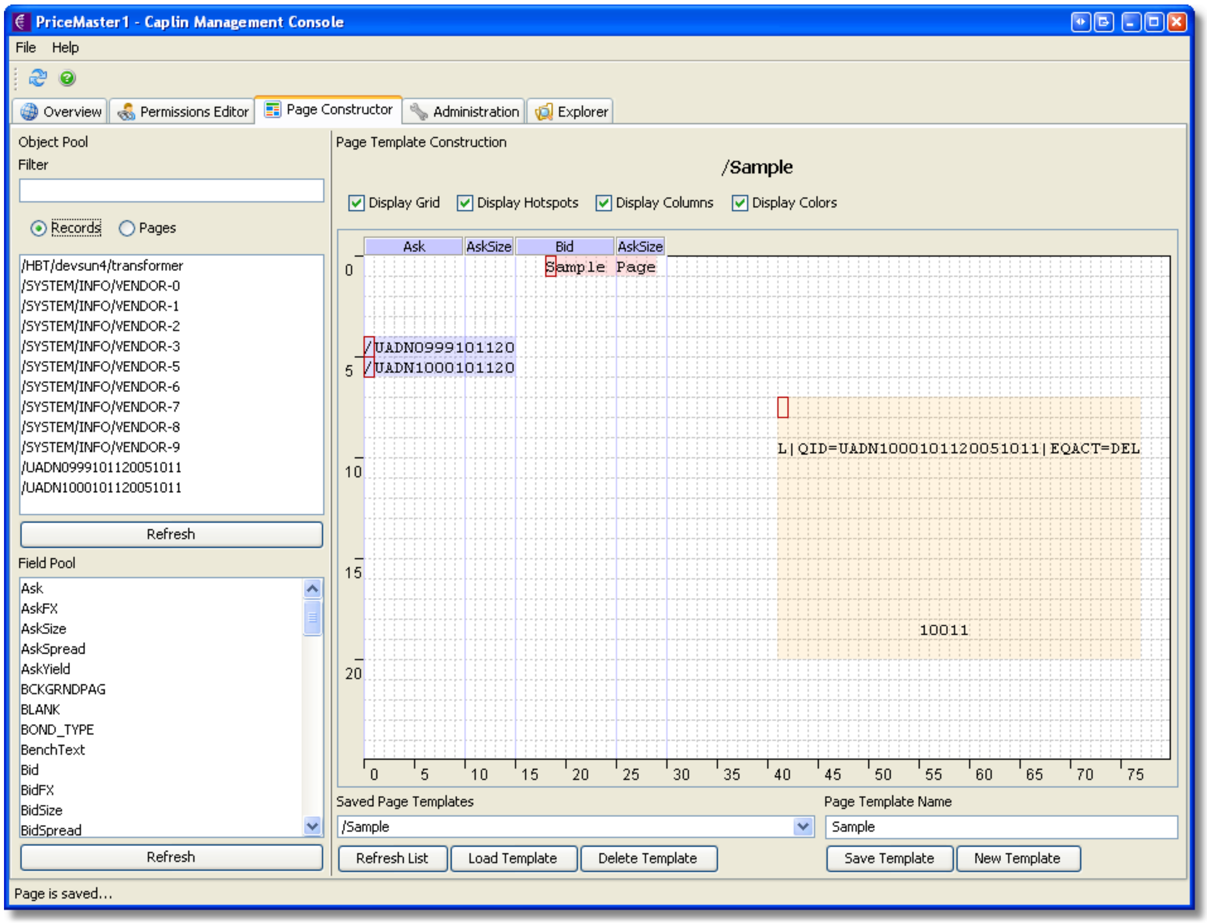
Permissioned Objects (All [✓]):

Object	Reut...	Bridg...	Trad...	FED-5	MLT...	MLT...	CQG...	TRS-9
/GCF<30_C-08/31_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-08/31_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-09/20_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-09/20_$_CHASE	☒	☒	✖	✓	☒	☒	☒	☒
/GCF<30_C-09/30_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-09/30_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-10/31_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-10/31_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-11/01_$_BONY	✖	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-11/01_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-11/30_$_BONY	☒	✖	☒	✓	☒	☒	☒	☒
/GCF<30_C-11/30_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-12/13_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-12/13_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-12/30_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-12/30_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-1W_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-1W_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-1Y_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-1Y_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-2W_$_BONY	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C-2W_$_CHASE	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C_$_500_LOT_B...	☒	☒	☒	✓	☒	☒	☒	☒
/GCF<30_C_$_500_LOT_C...	☒	☒	☒	✓	☒	☒	☒	☒

The PriceMaster Permissions Editor

PriceMaster Page Constructor

The Page Constructor enables pages to be built with any combination of the objects being received by PriceMaster. Objects are added to your page template simply by being dragged from the record pool. You can create columns in your template to represent the fields of the records which will be displayed on the page.



/Sample

Display Grid [✓] Display Hotspots [✓] Display Columns [✓] Display Colors [✓]

[✓ Records] Pages

/HBT/devsun4/transformer
/SYSTEM/INFO/VENDOR-0
/SYSTEM/INFO/VENDOR-1
/SYSTEM/INFO/VENDOR-2
/SYSTEM/INFO/VENDOR-3
/SYSTEM/INFO/VENDOR-5
/SYSTEM/INFO/VENDOR-6
/SYSTEM/INFO/VENDOR-7
/SYSTEM/INFO/VENDOR-8
/SYSTEM/INFO/VENDOR-9
/UADN0999101120051011
/UADN1000101120051011

Ask
AskFX
AskSize
AskSpread
AskYield
BCKGRNDPAG
BLANK
BOND_TYPE
BenchText
Bid
BidFX
BidSize
BidSpread

```
Sample Page
/UADN0999101120
/UADN1000101120
L|QID=UADN1000101120051011|EQACT=DEL
10011
```

Saved Page Templates: /Sample

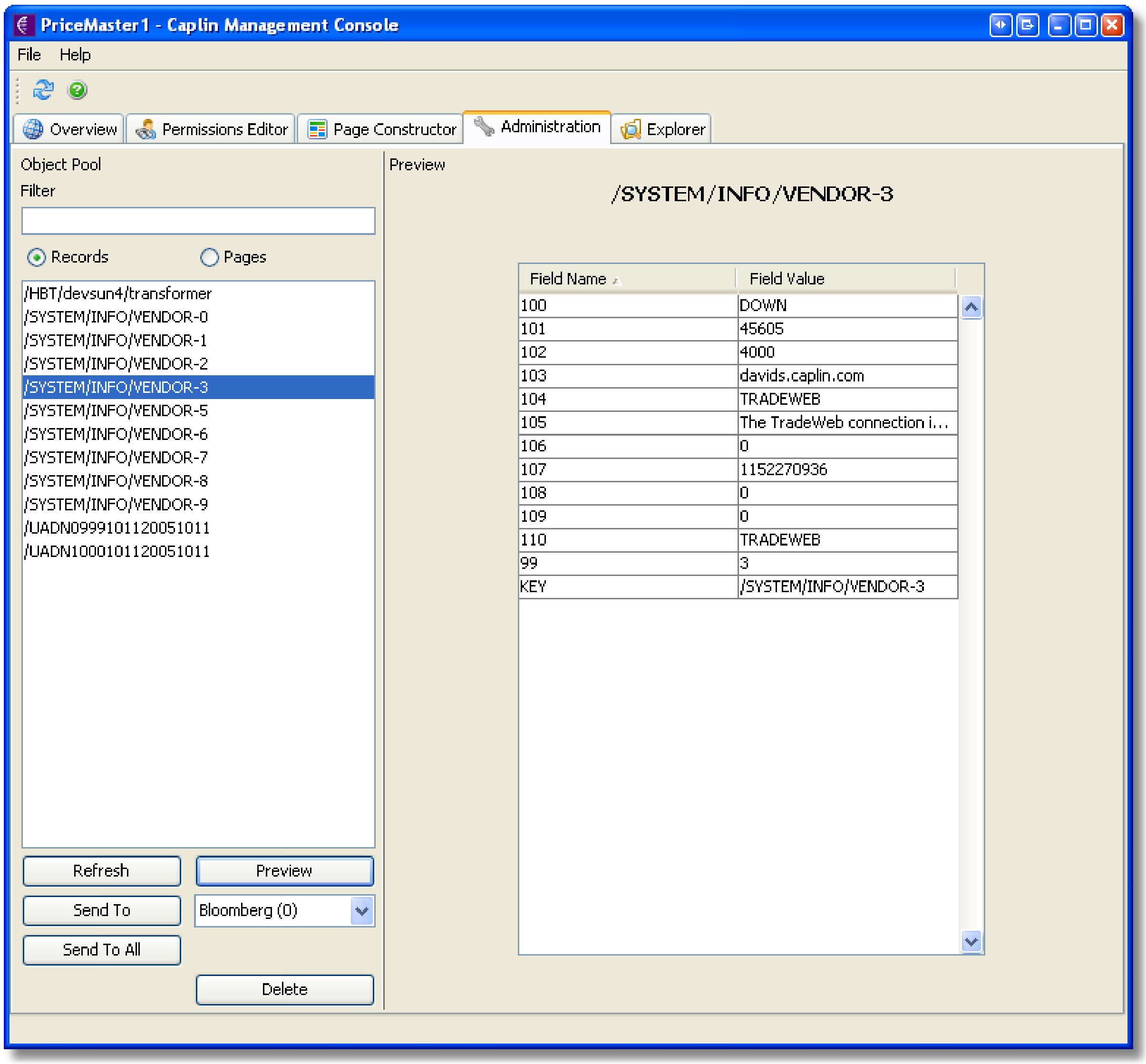
Page Template Name: Sample

Page is saved...

The PriceMaster Page Constructor

PriceMaster Administration

The Administration Module allows the current state of the cache to be viewed including individual record and page preview using real data. Additionally, objects can be deleted from the cache or they can be re-published to selected outputs. For example, you may find that a record or page is no longer being broadcast from one of the input data feeds. If this has happened and you know that one or more of your output datafeeds is in broadcast mode, then you may wish to delete that record or page from the Transformer's cache so that it does not continue to broadcast an empty record.



/HBT/devsun4/transformer
/SYSTEM/INFO/VENDOR-0
/SYSTEM/INFO/VENDOR-1
/SYSTEM/INFO/VENDOR-2
/SYSTEM/INFO/VENDOR-3
/SYSTEM/INFO/VENDOR-5
/SYSTEM/INFO/VENDOR-6
/SYSTEM/INFO/VENDOR-7
/SYSTEM/INFO/VENDOR-8
/SYSTEM/INFO/VENDOR-9
/UADN0999101120051011
/UADN1000101120051011

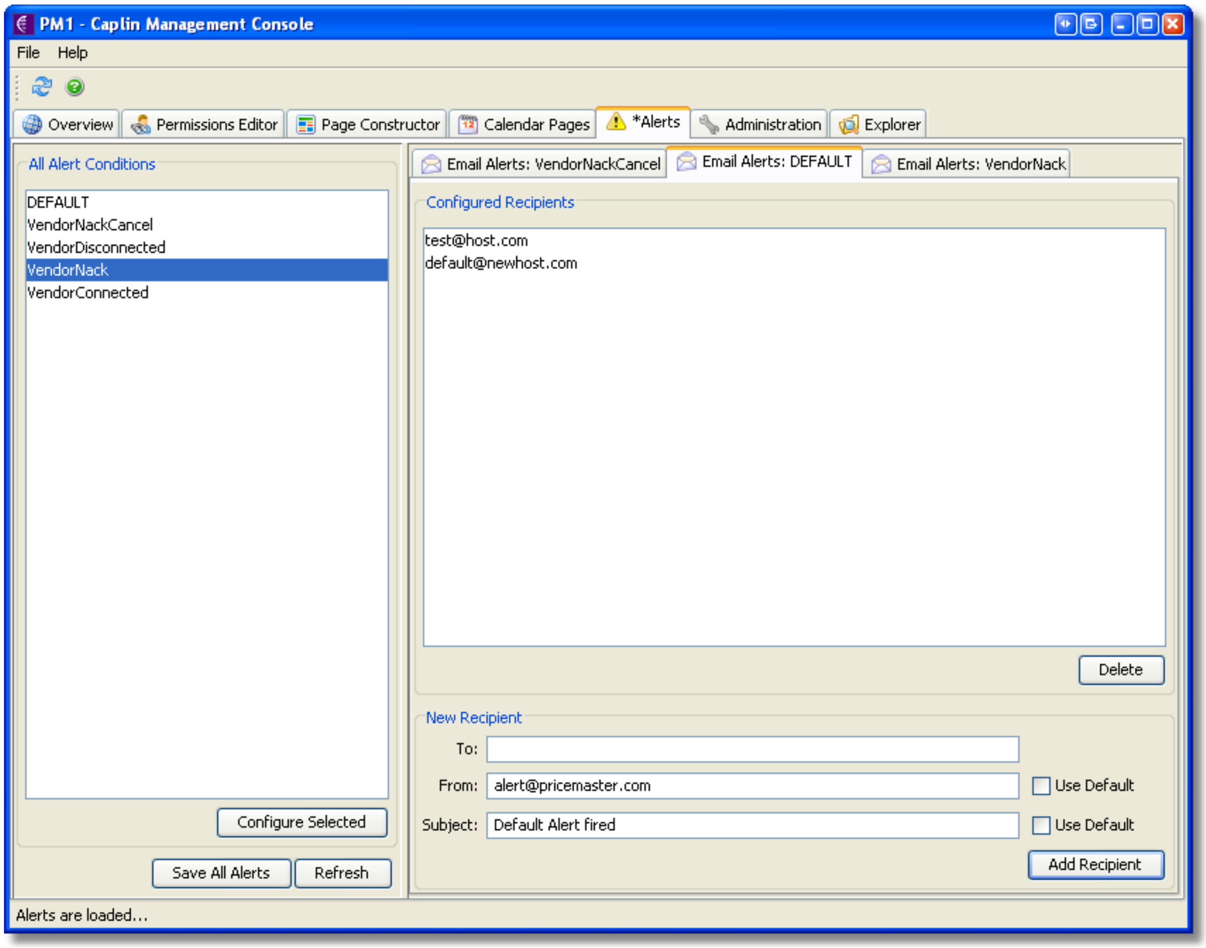
/SYSTEM/INFO/VENDOR-3

Field Name	Field Value
100	DOWN
101	45605
102	4000
103	davids.caplin.com
104	TRADEWEB
105	The TradeWeb connection i...
106	0
107	1152270936
108	0
109	0
110	TRADEWEB
99	3
KEY	/SYSTEM/INFO/VENDOR-3

The PriceMaster Administration view

Alerts Configuration Editor

The Alerts Configuration Editor is used to configure the actions to be taken when an alert is generated. It supports the sending of emails to multiple recipients when an alert is generated. The email subject and from address can also be configured. The DEFAULT alert condition is provided as a catch-all and can be used if the same email address is required for all alerts.



DEFAULT
VendorNackCancel
VendorDisconnected
VendorNack
VendorConnected

test@host.com
default@newhost.com

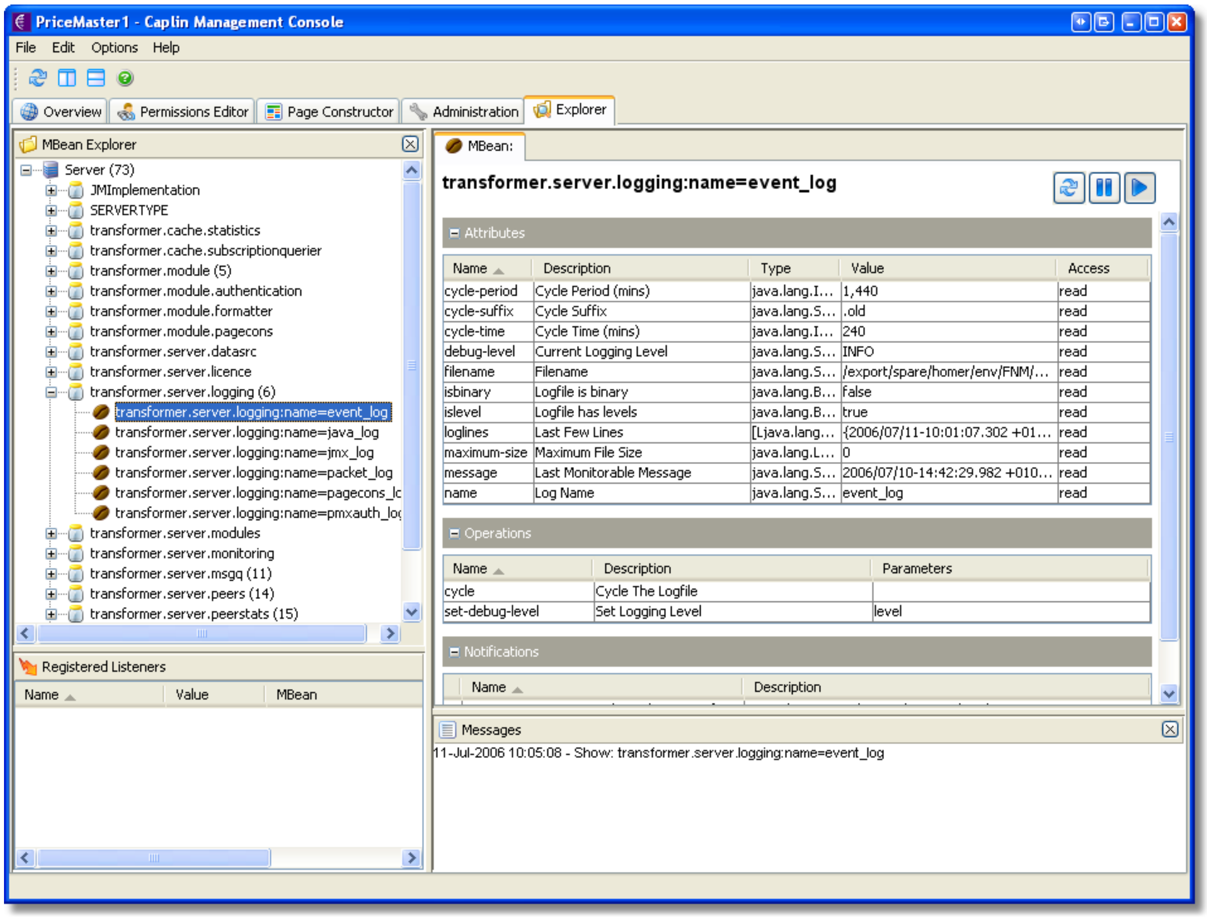
To:
From: alert@pricemaster.com
Subject: Default Alert fired

Alerts are loaded...

The Alerts Configuration Editor

Explorer View

The Explorer view is an advanced JMX browser that gives access to all the monitoring and management methods exposed by the PriceMaster. It is recommended for advanced users or troubleshooting only as some operations such as unload module or stop process may interrupt PriceMaster's operation. This view may be disabled for some installations.



Server (73)
- JMImplementation
- SERVERTYPE
- transformer.cache.statistics
- transformer.cache.subscriptionquerier
- transformer.module (5)
- transformer.module.authentication
- transformer.module.formatter
- transformer.module.pagecons
- transformer.server.datasrc
- transformer.server.licence
- transformer.server.logging (6)
  - transformer.server.logging:name=event_log
  - transformer.server.logging:name=java_log
  - transformer.server.logging:name=jmx_log
  - transformer.server.logging:name=packet_log
  - transformer.server.logging:name=pagecons_lc
  - transformer.server.logging:name=pmxauth_lo
- transformer.server.modules
- transformer.server.monitoring
- transformer.server.msgq (11)
- transformer.server.peers (14)
- transformer.server.peerstats (15)

Name	Value	MBean

transformer.server.logging:name=event_log

Name	Description	Type	Value	Access
cycle-period	Cycle Period (mins)	java.lang.I...	1,440	read
cycle-suffix	Cycle Suffix	java.lang.S...	.old	read
cycle-time	Cycle Time (mins)	java.lang.I...	240	read
debug-level	Current Logging Level	java.lang.S...	INFO	read
filename	Filename	java.lang.S...	/export/spare/homer/env/FNM/...	read
isbinary	Logfile is binary	java.lang.B...	false	read
islevel	Logfile has levels	java.lang.B...	true	read
loglines	Last Few Lines	[Ljava.lang....	{2006/07/11-10:01:07.302 +01...	read
maximum-size	Maximum File Size	java.lang.L...	0	read
message	Last Monitorable Message	java.lang.S...	2006/07/10-14:42:29.982 +010...	read
name	Log Name	java.lang.S...	event_log	read

Name	Description	Parameters
cycle	Cycle The Logfile	
set-debug-level	Set Logging Level	level

Name	Description

```
11-Jul-2006 10:05:08 - Show: transformer.server.logging:name=event_log
```

The Explorer view

Section IV

Installing PriceMaster

4.1 Creating an installation of PriceMaster

The first step in the installation of PriceMaster is to install the PriceMaster files and directories.

PriceMaster Archive Kit Naming

PriceMaster is delivered as a compressed archive for a specific server platform. The kit name will be similar to:

Linux: `PriceMaster-4.2.0-1-i686-pc-linux-gnu.tar.gz`
Solaris: `PriceMaster-4.2.0-1-sparc-sun-solaris2.8.tar.Z`

Some PriceMaster kits contain customer specific optional components and configuration, in which case the kit would be named to include a customer reference code.

Symbolic Linked Installation Structure

The recommended installation structure is to keep the extracted kit separate from the live PriceMaster installation. Symbolic links in the live PriceMaster installation are used to point to physical files within the kit. This is the recommended style of installation for Caplin products as it allows multiple configurations of the software to be used at the same time and also means that upgrading from one version to another is quick and easy. To achieve this the Unix *ln* command (link) is used, the usage of this command is shown below.

ln -s TargetFileOrDirName AliasFileorDirName

It is suggested that the -s option is used to make links symbolic. The *TargetFileOrDirName* is the name of the file / directory that is to be linked to and the *AliasFileOrDirName* the name by which the file / directory can be referenced.

Installation Steps

1. First choose a location on your filesystem for PriceMaster - this will be referred to as *$INSTALL_DIR* in this document. Then, create the directories where PriceMaster will be installed; the following directories are required:

- *kits/PriceMaster* - This directory will contain the PriceMaster installation kit,
- *PriceMaster1* - This directory will be used as the live PriceMaster installation but will use a link to the kit in the above directory.

In this example, *$INSTALL_DIR* is assumed to be */opt/caplin*. To create the directories change directory to *$INSTALL_DIR* and create the kits and installation folder:

```
$ cd /opt/caplin
$ mkdir –p kits/PriceMaster
$ mkdir –p PriceMaster1
```

Note: The *-p* option creates parent directories if necessary.

2. Copy the PriceMaster installation kit into the */kits/PriceMaster* directory. The example below copies the kit from */tmp* so this should be replaced with the current location.

```
$ cp /tmp/PriceMaster-4.2.0-1-i686-pc-linux-gnu.tar.gz  kits/PriceMaster
```

Note: The kit shown above is an example, the actual kit may not have this name because the version number may be different.

3. The installation kit is a compressed tar file, the next step is to un-compress and un-tar this file. From the *kits/PriceMaster* directory type the following.

On Linux use:

```
$ tar xzf PriceMaster-4.2.0-1-i686-pc-linux-gnu.tar.gz
```

On Solaris use:

```
$ uncompress PriceMaster-4.2.0-1-sparc-sun-solaris2.8.tar.Z
$ tar xf PriceMaster-4.2.0-1-sparc-sun-solaris2.8.tar
```

At this stage the *kits/PriceMaster* directory will contain the PriceMaster installation within a directory called *PriceMaster-4.2.0-1* (this will be the same name as the version of the file).

4. From the *kits/PriceMaster* directory, create a symbolic link to the PriceMaster installation directory. Use the following command to achieve this.

```
$ ln -s PriceMaster-4.2.0-1 latest
```

Having executed this command there will be two directories within the */kits/PriceMaster* directory; the first is the installation of the new PriceMaster, the second is a *latest* link to that installation. This can be seen by inspecting the output of ls -l, for example:

```
$ ls -l
drwxr-xr-x   4 tests s     512 Jul 10 07:27 .
drwxr-xr-x   3 tests s     512 Jul 10 07:27 ..
lrwxrwxrwx   1 tests s      30 Jul 10 07:27 latest -> PriceMaster-4.2.0-build.1
drwxr-xr-x  12 tests s     512 Jul 10 07:27 PriceMaster-4.2.0-build.1
```

5. Now change directories to the *PriceMaster1* directory so that the installed implementation of PriceMaster can be configured. First, navigate to the *PriceMaster1* directory and create logical links to some of the *../kits/PriceMaster/latest/* subdirectories using the following commands.

```
$ cd ../PriceMaster1
$ ln –s ../kits/PriceMaster/latest/bin bin
$ ln –s ../kits/PriceMaster/latest/doc doc
$ ln –s ../kits/PriceMaster/latest/lib lib
$ ln –s ../kits/PriceMaster/latest/xml_schemas xml_schemas
```

What has been achieved at this point are logical links to */latest* meaning that the contents of this directory will always point to the latest PriceMaster deployment. To change the live deployment the */latest* directory could simply be linked to another implementation in a different directory. The diagram below illustrates the links that have been made.

6. Next create the directory where the log files will be written: *var*. From the */PriceMaster1* directory execute the following command:

```
$ mkdir var
```

7. Finally copy the *etc, perm* and *templates* directory from the *latest* directory to this directory. These folders contain files that may be customized and adjusted by the local installation, and so should be copied locally.

```
$ cp –r ../kits/PriceMaster/latest/etc etc
$ cp –r ../kits/PriceMaster/latest/perm perm
$ cp –r ../kits/PriceMaster/latest/templates templates
```

Note: These folders do not use a symbolic link as they contain the config files that should not be overwritten by an upgrade. Also, having unlinked *etc*, *perm* and *templates* directories allows multiple configurations to be used for different instances of PriceMaster on the same server.

Listing the contents of the *PriceMaster1* directory should now show the following directories:

```
$ ls -l
total 9
lrwxrwxrwx  1 tests s   30 Jul 10 07:37 bin -> ../kits/PriceMaster/latest/bin
lrwxrwxrwx  1 tests s   30 Jul 10 07:37 doc -> ../kits/PriceMaster/latest/doc
drwxr-xr-x  2 tests s 1024 Jul 10 07:40 etc
lrwxrwxrwx  1 tests s   30 Jul 10 07:37 lib -> ../kits/PriceMaster/latest/lib
drwxr-xr-x  2 tests s  512 Jul 10 07:40 perm
drwxr-xr-x  2 tests s  512 Jul 10 07:40 templates
drwxr-xr-x  2 tests s  512 Jul 10 07:42 var
lrwxrwxrwx  1 tests s   38 Jul 10 07:37 xml_schemas -> ../kits/PriceMaster/latest/xml_schemas
```

- *bin* - Contains the binary programs for PriceMaster
- *doc* - Contains PriceMaster documentation and example programs
- *etc* - Contains start-up scripts and configuration files for PriceMaster
- *lib* - Contains the PriceMaster modules
- *perm* - Contains PriceMaster output permission configuration XML files
- *templates* - Contains PriceMaster page construction template XML files
- *var* - Contains the PriceMaster log files
- *xml_schemas* - Contain XML schemas used for PriceMaster configuration

Upgrading PriceMaster

To install a new version of PriceMaster simply:

- Stop PriceMaster
- Copy the kit archive file to the /kits/PriceMaster directory and un-compress it (using steps 2 and 3)
- Change the symbolic latest link to point at the new implementation (using step 4)
- Copy any new or updated configuration that is required to the installation folder (using step 5 or taking individual files)
- Start the newly upgraded PriceMaster

Tip: When PriceMaster is upgraded the etc, perm and templates directories containing the configuration are not overwritten, despite this it is a good idea to take a backup of these directories before commenting an upgrade.

4.2 Configuring PriceMaster for your environment

Once the basic PriceMaster installation is created, it must be configured to run in your environment. To do this, the *pricemaster* control script needs to have the root location modified to point to the application's installation directory and the Java JVM library location and executable location must be configured.

1. Configuring the PriceMaster control script

PriceMaster is controlled with the script *pricemaster* which can be found in the *etc* directory. Edit this file to change the variable *PRICEMASTER_ROOT* to point to the root directory of the PriceMaster installation (in this case *$INSTALL_DIR/PriceMaster1*). This file should be edited using an editor such as vi or emacs, for example:

```
vi etc/pricemaster
```

The extract below shows the *etc/pricemaster* file before and after it has been edited to adjust the root location.

Before:

```
...
if [ -z "$PRICEMASTER_ROOT" ]; then
    PRICEMASTER_ROOT=/opt/PriceMaster
fi
...
```

After:

```
...
if [ -z "$PRICEMASTER_ROOT" ]; then
    PRICEMASTER_ROOT=/opt/caplin/PriceMaster1
fi
...
```

2. Configuring the Java JVM location

PriceMaster uses a Java Virtual Machine (JVM) to provide it's management and monitoring capabilities and allow the Enterprise Management Console to configure and control it. A 1.5.0 or newer JVM is required to run PriceMaster.

When the server process is started it dynamically loads a JVM using the configuration defined in the file *etc/java.conf*. This configuration file must contain the location of an installed 1.5.0 or newer JVM. The configuration option *jvm-location* must be modified to point to the location of the libjvm.so library module file provided by the JVM installation. This file is typically located in *$JAVA_HOME/lib/<platform>/server/libjvm.so.*

The following example assumes the JVM library module is located in */opt/java/jdk/jre/lib/sparc/server/libjvm.so.*

The extract below shows the *etc/java.conf* file before and after it has been edited to adjust the JVM library module location.

Before:

```
...
# Set jvm-location as a fully qualified pathname to the preferred JVM
#
jvm-location       /usr/local/jdk/jre/lib/sparc/libjvm.so
...
```

After:

```
...
# Set jvm-location as a fully qualified pathname to the preferred JVM
#
jvm-location       /opt/java/jdk/jre/lib/sparc/server/libjvm.so
...
```

Some of the Java-based output handlers (for example TradeWeb) may also need the java location to be set. If set, the environment variable *$JAVA* will be used to run the java command. This environment variable should point to the java executable. Alternatively, this can be set in the PriceMaster control script (*pricemaster* which can be found in the *etc* directory)

The following example assumes the Java executable located at */opt/java/jdk/jre/bin/java*.

The extract below shows the *etc/pricemaster* file before and after it has been edited to adjust the location of the java executable.

Before:

```
...
if [ -z "$JAVA" ]; then
    JAVA=/usr/local/jdk/jre/bin/java
    export JAVA
fi
...
```

After:

```
...
if [ -z "$JAVA" ]; then
    JAVA=/opt/java/jdk/jre/bin/java
    export JAVA
fi
...
```

4.3 Installing the PriceMaster License

In order for PriceMaster to operate correctly a valid license file must be installed. A license is usually supplied by Caplin separately from the software installation kit.

Follow these steps to install / upgrade the license file:

1. Backup the existing license (if you have one).

The following example assumes you are in the PriceMaster live installation directory (e.g. *$INSTALL_DIR/PriceMaster1*)

```
$ cp etc/license-pricemaster.conf  etc/license-pricemaster.conf.old
```

2. Copy the new license to the etc as license-transformer.conf.

The following example assumes your new license is named LIC_LC_1006.PM1.conf and is in the /tmp folder

```
$ cp /tmp/LIC_LC_1006.PM1.conf etc/license-pricemaster.conf
```

Note: Ensure that PriceMaster is not running before attempting a licence installation.

Note: PriceMaster will only run for 30 minutes by default if a valid licence is not installed.

Section V

Controlling and Monitoring PriceMaster

5.1 Starting and stopping the PriceMaster components

The PriceMaster control script is used for starting and stopping the PriceMaster in its entirety or for controlling the individual component processes (for example, stopping a particular output handler). The PriceMaster control script is called *pricemaster* and is located in the *etc* directory of the installation.

The control script uses files within the *bin* and *etc* directories, it uses the *PRICEMASTER_ROOT* variable (as configured in the previous section) to locate these files.

The following examples assumes you have installed and configured PriceMaster and are in the PriceMaster live installation directory (e.g. *$INSTALL_DIR/PriceMaster1*)

Starting the PriceMaster

To start the PriceMaster run the PriceMaster control script with the parameter *start:*

```
$ etc/pricemaster start
```

If successful, the script should output details of the transformer and each input and output that has been started, for example:

```
Starting Transformer
Starting input 30 ddpsrc
Starting input 31 ddpsrc
Starting output umdpsink1 - Reuters-1
Starting output umdpsink2 - Bridge-2
Starting output umdpsink2 - Telerate-3
transformer: License will expire on Wed Sep 27 01:00:00 2006
```

Tip: If you see an error message like:

```
etc/pricemaster: line 82: /opt/PriceMaster/bin/seq: No such file or
directory
```

when starting PriceMaster, then this means that the PRICEMASTER_ROOT variable in the pricemaster control script is not set correctly.

Tip: If you see an error message like:

```
transformer: Couldn't open JVM at /usr/local/jdk/jre/lib/sparc/libjvm.so -
/usr/local/jdk/jre/lib/sparc/libjvm.so: cannot open shared object file: No
such file or directory - exiting
```

when starting PriceMaster, then this means that the JVM location has not been configured correctly. See the previous section for details on this configuration.

Stopping the PriceMaster

To stop the PriceMaster run the PriceMaster control script with the parameter *stop:*

```
$ etc/pricemaster stop
```

If successful, the script should output details of the transformer and each input and output that has been stopped, for example:

```
Killing process transformer with pid 7686
Waiting for transformer 7686
Killing process Reuters-1 with pid 7804
Killing process Bridge-2 with pid 7814
Waiting for Bridge-2 7814
Killing process Telerate-3 with pid 7814
Killing process ddpsrc30 with pid 7730
Killing process ddpsrc31 with pid 7743
```

Starting and stopping individual component processes

You may need to stop or start individual processes or groups of processes, for example to update the transformer configuration or to re-start an individual output for a vendor connection configuration change. The control script supports the following command parameters:

Command parameter	***Description***
start / stop	Starts or stops the entire PriceMaster - the transformer core process and all configured input and output processes.
start-transformer / stop-transformer	Starts or stops the main PriceMaster transformer process.
start-inputs / stop-inputs	Starts or stops all input processes
start-outputs / stop-outputs	Starts or stops all output processes
start-input *n type* / stop-input *n type*	Starts or stops the input process number *n* of the type specified by *type*. The type refers to the type of the input DataSource adapter, for example ddpsrc, omd2src, xlpsrc, etc.
start-output *n type* / stop-output *n type*	Starts or stops the output process number *n* of the type specified by *type*. The type refers to the vendor connection type of the output, for example umdpsink, mpfsink, TradeWebSource, etc.

For example, to start output *2* of type *umdpsink*:

```
$ etc/pricemaster start-output 2 umdpsink
```

If successful, the script should output details of the output that has been started (including its display name, that may vary depending on configuration - in this case **Bridge-2**):

```
Starting output umdpsink2 - Bridge-2
```

5.2 Using the Enterprise Management Console

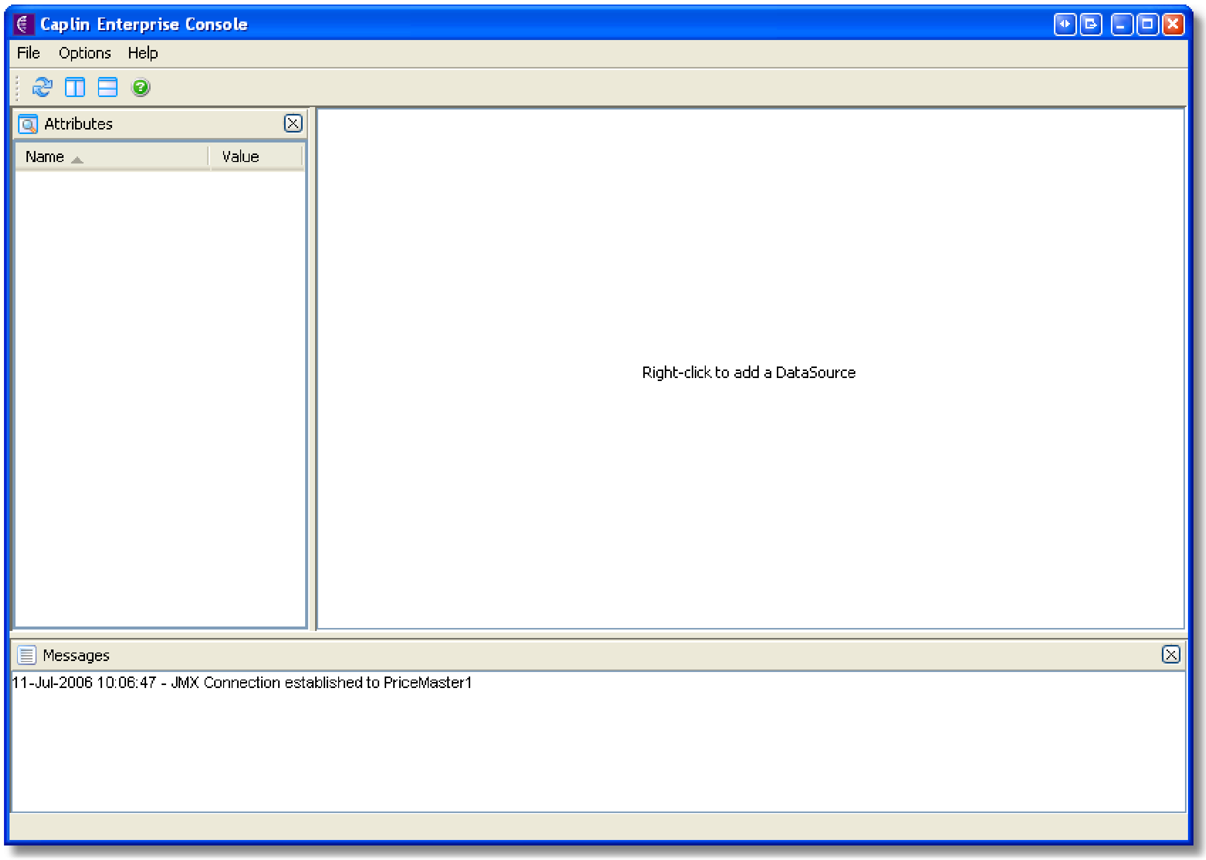
The Caplin Enterprise Management Console is supplied as a kit containing an executable Java Archive (JAR), required libraries and documentation and code examples detailing how to customize the console. It requires Java 1.5.0 or newer.

The following guide will show how to get started with the Enterprise Managment Console and connect it to your PriceMaster. Refer to the documentation included in the kit for full details on server and client configuration, customization of the console and an overview of the JMX monitoring API that can be used to connect other management tools to PriceMaster. The application also includes an extensive online help guide to provide assistance with the operation of each of the views.

The application is contained within the executable jar file *emc.jar*. On windows, this can usually be executed directly by double-clicking directly on the file. This should use java to launch the application. If java is not associated with the .jar filetype, or if you are running the Enterprise Management Console on a non-windows platform, it can be executed from the command line using the following command:

```
$ java –jar emc.jar
```

When the console is loaded for the first time, you will see the the following screen:

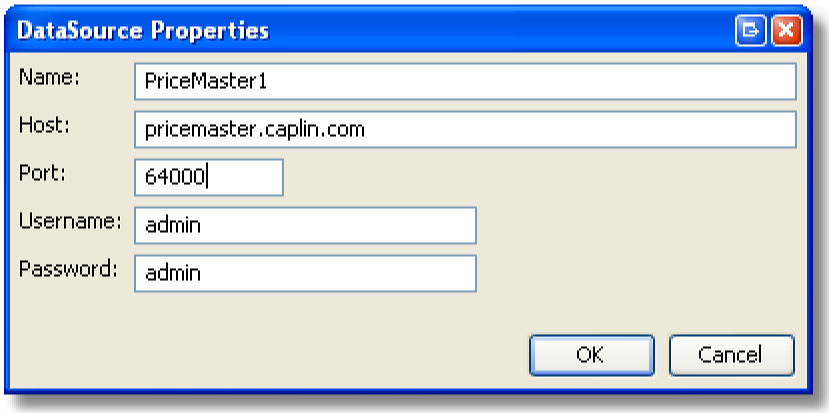


```
11-Jul-2006 10:06:47 - JMX Connection established to PriceMaster1
```

Initial Enterprise Console screen

To monitor and manage the PriceMaster, you should add it as a DataSource to the Enterprise screen by right clicking and choosing
Add New DataSource... The following dialog will appear, where you should enter the details of your PriceMaster server host and port. The default port is 64000, if you have changed this in your server

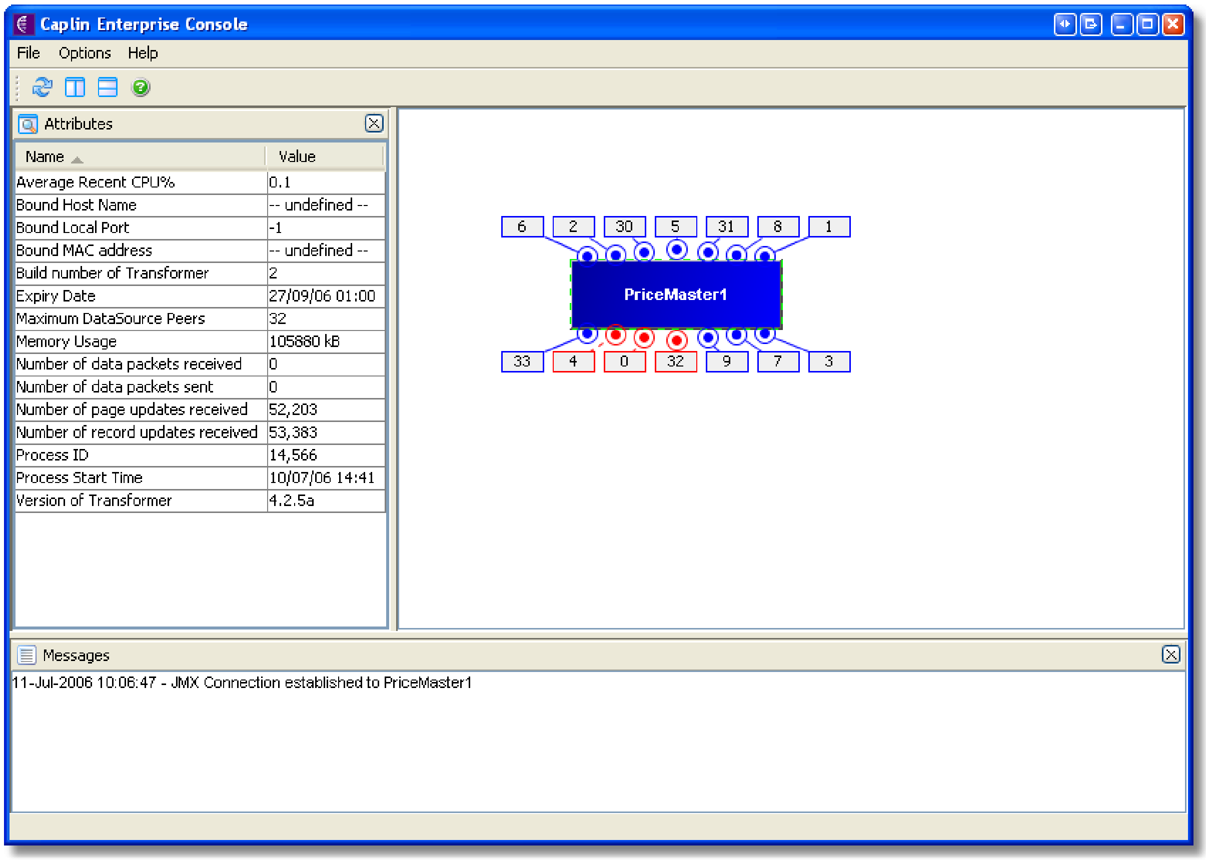
configuration then you should adjust the setting accordingly:



Name:	PriceMaster1
Host:	pricemaster.caplin.com
Port:	64000
Username:	admin
Password:	admin

New DataSource properties dialog

A connection will be made to the PriceMaster and the Enterprise screen should now look similar to the following image. When the PriceMaster is selected (by clicking on the blue box) a number of high-level information attributes about the PriceMaster instance will be displayed in the left panel.



Name	Value
Average Recent CPU%	0.1
Bound Host Name	-- undefined --
Bound Local Port	-1
Bound MAC address	-- undefined --
Build number of Transformer	2
Expiry Date	27/09/06 01:00
Maximum DataSource Peers	32
Memory Usage	105880 kB
Number of data packets received	0
Number of data packets sent	0
Number of page updates received	52,203
Number of record updates received	53,383
Process ID	14,566
Process Start Time	10/07/06 14:41
Version of Transformer	4.2.5a

```
11-Jul-2006 10:06:47 - JMX Connection established to PriceMaster1
```

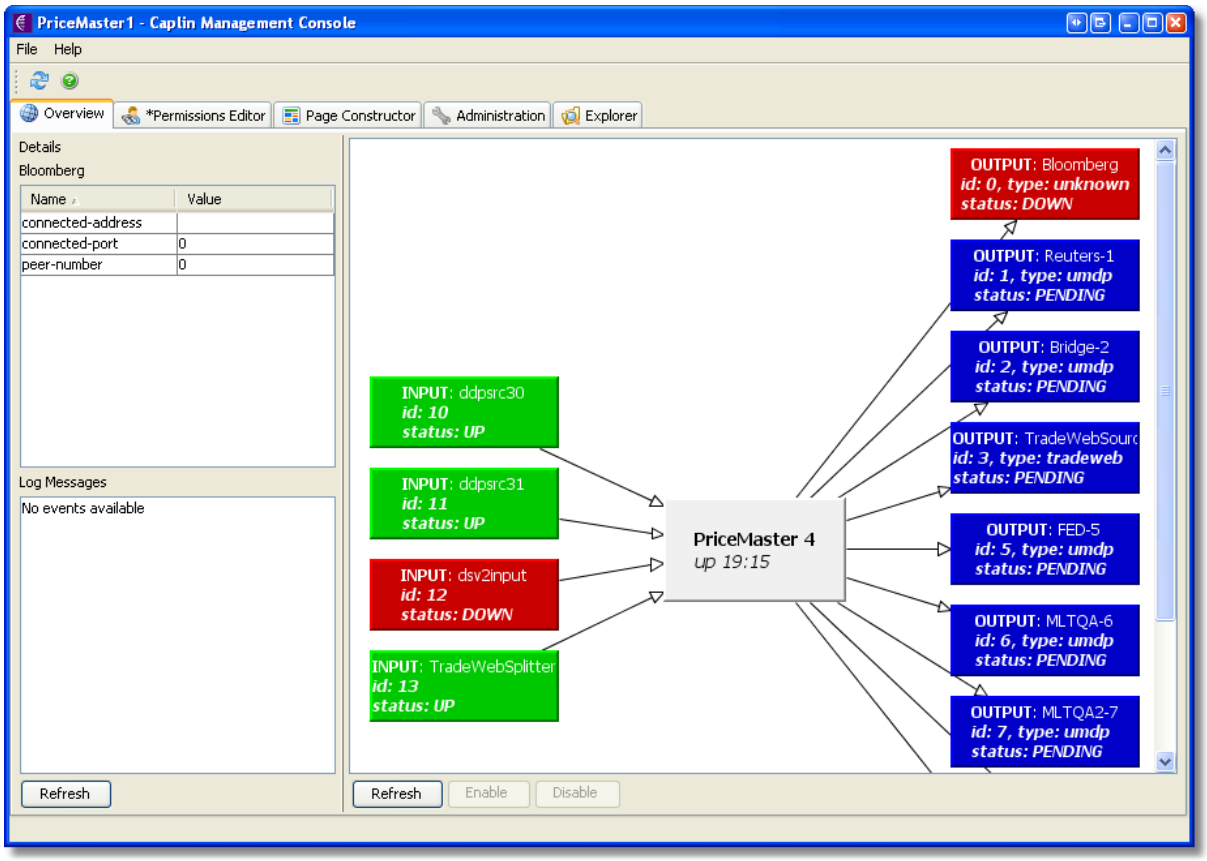
Enterprise Console showing a configured and connected PriceMaster instance

Tip: If the PriceMaster1 box is red, and you see an error message in the Messages pane like:

```
11-Jul-2006 11:14:35 - Error loading datasource details: PriceMaster1
11-Jul-2006 11:14:35 - Failed to retrieve RMIServer stub:
```

```
javax.naming.CommunicationException [Root exception is
java.rmi.NoSuchObjectException: no such object in table]
```
this means the console has not been able to connect to the PriceMaster. Check the host and port settings are correct and that the PriceMaster server is running.

To access the PriceMaster specific Console, double-click on the PriceMaster box, or right-click and choose *Open Console*. The PriceMaster Console will then be shown, allowing monitoring, control and configuration of the PriceMaster instance. Please refer to the online help for full details on using the PriceMaster Console.


PriceMaster1 - Caplin Management Console
File
Help
Overview
*Permissions Editor
Page Constructor
Administration
Explorer
Details
Bloomberg
OUTPUT: Bloomberg id: 0, type: unknown status: DOWN
OUTPUT: Reuters-1 id: 1, type: umdp status: PENDING
OUTPUT: Bridge-2 id: 2, type: umdp status: PENDING
OUTPUT: TradeWebSourc id: 3, type: tradeweb status: PENDING
OUTPUT: FED-5 id: 5, type: umdp status: PENDING
OUTPUT: MLTQA-6 id: 6, type: umdp status: PENDING
OUTPUT: MLTQA2-7 id: 7, type: umdp status: PENDING
INPUT: ddpsrc30 id: 10 status: UP
INPUT: ddpsrc31 id: 11 status: UP
INPUT: dsv2input id: 12 status: DOWN
INPUT: TradeWebSplitter id: 13 status: UP
PriceMaster 4 up 19:15
Refresh
Refresh
Enable
Disable

Name	Value
connected-address	
connected-port	0
peer-number	0

Log Messages

No events available

PriceMaster Console Overview

Section VI

Adding inputs and outputs

6.1 Input and Output processes

All inputs and outputs are DataSource peers of the Transformer process. They run as separate processes and for each input and output there is a corresponding **add-peer / end-peer** configuration group in the Transformer's configuration file (*transformer.conf*).

What is an output?

All outputs are separate DataSource processes that covert the DataSource protocol to the onward vendor feed and / or provide status information and data acknowledgments. They are responsible for delivering the data provided by PriceMaster to the vendor or onward distribution system using the appropriate protocol. They are provided with the PriceMaster kit, run on the same physical machine as the main Transformer process and are started using the main *etc/pricemaster* control script.

What is an input?

Some input processes are provided as separate DataSource processes that convert a feed to the DataSource protocol and may be provided with the PriceMaster kit. Other inputs are custom built applications / DataSource handlers written using Caplin's C or Java DataSource SDK. They may or may not run on the same physical machine as the main Transformer process and may not be started using the *etc/pricemaster* control script.

6.2 Adding an Input

Input peers are usually identified with peer id 30 and above. It is usually beneficial maintain sequential numbering for input peers starting from 30.

Any DataSource that uses Caplin's C or Java DSDK can be used as a DataSource input to PriceMaster. The standard DataSource peer configuration should be used to configure that DataSource to connect to the Transformer process in PriceMaster.

Note: PriceMaster only supports input DataSources that use the broadcast data paradigm - they should not require a request before providing data.

To add a new generic DataSource input:

Configure the DataSource to connect to PriceMaster

The configuration of the input DataSource will depend on whether it is a Java or C DataSource. The following configuration excerpts show how to configure

Java DataSource:

```
  <peer>
    <local id="34"/>
    <remote id="100"/>
    <destination address="pricemaster.company.com" port="26000"/>
  </peer>
```

C DataSource:

```
add-peer
    addr            pricemaster.mycompany.com
    port            26000
    local-id        34
    remote-id       100
end-peer
```

Add the input to the transformer's configuration

- Add the new input to etc/transformer.conf by creating a new add-peer / end-peer configuration group.
- The **remote-type** should be 0. This indicates that the input is a broadcast DataSource.
- The **remote-id** should match the local id in the input DataSource's configuration.
- The **remote-name** is a mnemonic for your convenience, and will appear on the PriceMaster 4 console in the overview diagram.

```
add-peer
    remote-id       34
    remote-type     0
    remote-name     TestInput
end-peer
```

Start the input

The input process will need to be restarted if its peer configuration has been changed.

The Transformer process will also need to be restarted to allow the new input peer to connect. The Transformer process can be restarted without restarting the other PriceMaster processes by issuing the following commands:

```
$ etc/pricemaster stop-transformer
$ etc/pricemaster start-transformer
```

6.3 Adding an Output

Output peers are normally identified with a peer id in the range 0 - 30. It is usually beneficial maintain sequential numbering starting from 0.

To add a new UMDP (MarketLink) output:

Adding a new UMDP output requires the creation of the configuration for a new umdpsink process and the addition of a peer configuration to the *etc/transformer.conf* file.

Create a new umdpsink configuration file:

- ❖ Copy *etc/umdpsink1.conf* and rename it, replacing the "1" with a number representing the new peer number (typically one higher than your current highest output peer).
- ❖ Change the **umdp-listen-port** option to the port that the will listen for the new vendor connection.
- ❖ Change the **local-id** option to the new peer number.

Tip: All peer numbers must be unique across all types of output handler

```
umdp-listen-port  10017

add-peer
    addr          localhost
    port          26000
    local-id      7
end-peer

client-name       Telerate
```

Add the output to the transformer's configuration

- ❖ Add the new output to *etc/transformer.conf* by creating a new **add-peer / end-peer** configuration group.
- ❖ The **remote-id** should match the number of the new *etc/umdpsink[n].conf* file you have created.
- ❖ The **local-type** should always be 1 - this indicates that the output handler can request data from the Transformer.
- ❖ The **remote-name** is a mnemonic for your convenience, and will appear on the PriceMaster 4 console in the overview diagram.

```
add-peer
    remote-id     7
    local-type    1
    remote-name   Telerate
end-peer
```

Start the output

Once the output is added it will be automatically started the next time the PriceMaster is started. It is possible to manually start the output and Transformer process without restarting the other PriceMaster processes by issuing the following commands, where n is the newly added output peer number:

```
$ etc/pricemaster start-output n umdpsink
$ etc/pricemaster stop-transformer
$ etc/pricemaster start-transformer
```

To add a new MPF (Bloomberg) output:

Adding a new MPF output is similar to adding a UMDP output. It requires the creation of the configuration for a new mpfsink process and the addition of a peer configuration to the transformer. conf.

Create a new mpfsink configuration file:

- ❖ Copy *etc/mpfsink0.conf* and rename it, replacing the “0” with a number representing the new peer number (typically one higher than your current highest output peer).
- ❖ Change the **mpf-contrib-server** and **mpf-contrib-port** options in to set the server and port that the new mpfsink instance should connect to.
- ❖ Change the **local-id** option to the new peer number.

```
debug-level         INFO

mpf-contrib-server  284.346.23.65
mpf-contrib-port    10034
city-code           LON
broker-code         CPLN

add-peer
    addr           localhost
    port           26000
    local-id       10
end-peer

client-name       Bloomberg2
```

Add the output to the transformer's configuration

- ❖ Add the new output to *etc/transformer.conf* by creating a new **add-peer / end-peer** configuration group.
- ❖ The **remote-id** should match the number of the new *etc/mpfsink[n].conf* file you have created.
- ❖ The **local-type** should always be 1 - this indicates that the output handler can request

data from the Transformer

- The **remote-name** is a mnemonic for your convenience, and will appear on the PriceMaster 4 console in the overview diagram.

```
add-peer
    remote-id     10
    local-type    1
    remote-name   Bloomberg2
end-peer
```

Start the output

Once the output is added it will be automatically started the next time the PriceMaster is started. It is possible to manually start the output and Transformer process without restarting the other PriceMaster processes by issuing the following commands, where n is the newly added output peer number:

```
$ etc/pricemaster start-output n mpfsink
$ etc/pricemaster stop-transformer
$ etc/pricemaster start-transformer
```

Section VII

Configuration Reference

7.1 Configuration Files

PriceMaster uses files within the *etc* directory to store its configuration. Some of these files are static and are read by PriceMaster on startup. Other files are dynamic and may change during operation as a result of a configuration adjustment made using the PriceMaster console.

Here follows a list of each configuration file and its use. Some installations may not include all of these files, depending on the inputs / outputs and modules present.

Name	***Type***	***Description***
alert-definitions.xml	dynamic	Configuration of the available alert types
alerter.conf	static	Static configuration for the alerter module
alerts	directory	Used to store backup/history of alerts configuration
alerts.xml	dynamic	Defines the actions to be followed when an alert is received (should not be manually edited)
calendarpages	directory	Used to store backup/history of calendar pages configuration
calendarpages.xml	dynamic	Defines which page stems are calendar pages (should not be manually edited)
cluster-primary.conf	dynamic	Definition of current cluster node status for persistent mode clustering (should not be manually edited)
cluster.conf	static	Defines the cluster (ftd) arrangement and control various options regarding replication of data
ddpsrc[n].conf	static	Configuration for the DSv1v2 converter input(s)
fields-bloomberg.conf	static	Fields used for Bloomberg output handler (mpfsink)
fields.conf	static	Configuration of field lists used by the Transformer
Fields.xml	static	Field configuration information used by the Java components
java.conf	static	Configuration of the in-process JVM used by the Transformer
jmx.conf	static	Configuration of the JMX plugin used by the Transformer
license-pricemaster.conf	license file	Licence for PriceMaster

mpfsink[n].conf	static	Configuration for the MPF (Bloomberg) output(s)
pagecons.conf	static	Static configuration used by the Page Construction Module
pmxauth.conf	static	Static configuration used by the Permissioning Module
pricemaster	startup script	Main startup script
transformer.conf	static	Configuration for the Transformer
umdpsinkX.conf	static	Configuration for the UMDP (MarketLink) output(s)

Where [n] corresponds to a the input/output number (in the range 0 - 63).

7.2 Alerter Module

The alerter module is responsible for generating alerts via email or to a file. This is triggered when DataSource alert messages are received to indicate that an alert condition has been met. The individual PriceMaster components generate DataSource alert messages for conditions such as vendor connections and disconnections and vendors not receiving data, depending on configuration.

The precise alert conditions available will depend on the components included in the PriceMaster installation and can be found on the Alerts tab in the PriceMaster console.

The alerter module is loaded using the `add-module alerter` configuration option in *transformer.conf*

Configuration Options

The following options can be used to configure the static configuration for the alerter module. These options can be changed in the *etc/alerter.conf* file and will be read on startup of the PriceMaster.

Tip: Runtime configuration of the alert actions (like which email addresses to notify when an alert is raised) is carried out using the alert tab of the PriceMaster console.

config-dir

Type:	String
Default Value:	%r/etc
Required:	No

The directory that stores the runtime configuration files for the alerts (alerts.xml and alert-definitions. xml). The files in this directory are read on startup and also when they are updated following changes made to the alerts configuration using the PriceMaster console.

Example:

```
config-dir    /etc/pm4/alerts
```

default-alert-action

Type:	String
Default Value:	DEFAULT
Required:	No

The name of the action to use if there are no matching configured actions for an alert message that has been received. This can be used as a catch-all or global configuration for alerts where no specific configuration is needed.

Example:

```
default-alert-action    catch-all
```

smtp-server

Type:	String
Default Value:	N/A
Required:	No

The email server that will be used to send email alerts. The provided value should to be a hostname or ip address that points to a valid smtp server which can accept mail. If this option is not provided then email alerts will be disabled.

Example:

```
smtp-server    mailserver.hostname.com
```

smtp-port

Type:	Integer
Default Value:	25
Required:	No

The port number of the email server that will be used to send email alerts. In most cases this will not need to change.

Example:

```
smtp-port    1025
```

email-from-address

Type:	String
Default Value:	alert@[hostname]
Required:	No

The default from address to use if the email alert configuration doesn't specify one. The default value will be alert@[hostname] where [hostname] is the host name of the machine or the value configured by the local-hostname option.

Example:

```
email-from-address    alert@caplinpricemaster
```

local-hostname

Type:	String
Default Value:	N/A
Required:	No

The host identifier to use when communicating with the smtp server. This may be required if the system-supplied hostname (for example that returned by the *hostname* command) is not adequate for communication with the smtp server.

Example:

```
local-hostname    pm1.company.com
```

alert-directory

Type:	String
Default Value:	N/A
Required:	No

The name of the directory to use for disc alert logging. If not present then disc alert logging will be disabled.

Example:

```
alert-directory    %r/alerts
```

log-file

Type:	String
Default Value:	alerter.log
Required:	No

The file that should be used for logging messages generated by the alerter module.

Example:

```
log-file    alerter-test.log
```

log-level

Type:	String
Default Value:	INFO
Required:	No

The maximum level of messages that will be logged. This value should be one of ERROR, WARN, NOTIFY, INFO, DEBUG. In normal operation the INFO level should be used to avoid generating excessively large logs. For troubleshooting purposes the DEBUG level should be used and will output the most detailed information.

Example:

```
log-level    DEBUG
```

7.3 Cluster Module

The cluster module is responsible for clustering data and / or configuration between one or more PriceMaster instances. It is not required for a PriceMaster installation, but is useful when multiple identical instances of PriceMaster are required for fault tolerance and failover.

PriceMaster clustering uses the concept of nodes. Each node (i.e. PriceMaster instance) is either a primary or a secondary node. There is only ever one primary node per cluster, all other nodes in the cluster are secondary nodes. Depending on the configuration of the cluster module the primary node is chosen either by a preferred node in configuration, is dynamic based on process uptime or is manually configured.

Cluster nodes are configured to connect to all other nodes in the cluster (one-to-many connection paradigm). Each node is configured using the add-cluster-node configuration group.

The cluster module is loaded using the `add-module cluster` configuration option in *transformer.conf*

Configuration Options

The following options can be used to configure the cluster module. These options can be changed in the *etc/cluster.conf* file and will be read on startup of the PriceMaster.

add-cluster-node / end-cluster-node

This configuration group is used to define the nodes in the cluster. There should be one instance of this group for each node in the cluster. The `cluster-index` configuration option is used to indicate which of these configured nodes identifies this node.

Tip: This configuration section should be identical for each node in the cluster.

The sub-options in this configuration group are:

Name	***Type***	***Default***	***Required***	***Description***
cluster-addr	String	127.0.0.1	Yes	The hostname or ip address of the node in the cluster
cluster	Integer	N/A	Yes	The port to either connect to or listen on. If this node is referenced by the cluster-index configuration option then this is a listen port, otherwise it is used for outbound connection

Example:

```
add-cluster-node
    cluster-addr    pm2.company.com
    cluster-port    14001
end-cluster-node
```

cluster-index

Type:	Integer
Default Value:	N/A
Required:	Yes

The cluster index for this node. Must be unique in the cluster (for example, a cluster containing 2 nodes would use 0 and 1 for this option in each node respectively).

Example:

```
cluster-index 0
```

primary-method

Type:	String
Default Value:	order
Required:	No

Defines the method used by the cluster to determine which node is primary on startup of a new node. the possible values for this configuration option are:

- **order** - this is the default method and means that the first cluster node in the configuration file will always be the primary node, if available.
- **uptime** - this method will make the node that has been up for longest primary.
- **persisted** - this method works in tandem with the primary-config-file option. It persists the state of the nodes so that if they are shutdown and restarted the primary node will not change.

Example:

```
primary-method persisted
```

primary-config-file

Type:	String
Default Value:	%r/etc/cluster-primary.conf
Required:	No

If using the **persisted** `primary-method` option, this file is used to persist the current state of the node. This file will be dynamically written any time the primary / secondary state of a node changes. The file will either contain nothing, or the single option `force-primary`.

Note: Upon initial installation, if using the **persisted** `primary-method` option, the desired initial primary node should have this file manually edited to include the `force-primary` option. This will ensure it starts as the primary node. Subsequently this file should not be manually edited as the cluster nodes will use it to automatically persist their state.

Example:

```
primary-config-file %r/etc/cluster-state.conf
```

force-primary

Type:	String
Default Value:	FALSE
Required:	No

If using the **persisted** `primary-method` option, this option may be present in the file referenced by that option. It indicates that the node is the primary node.

Example:

```
primary-config-file %r/etc/cluster-state.conf
```

Note: This option should not be manually inserted into the main configuration file.

connect-sync

Type:	String
Default Value:	N/A
Required:	No

Pattern match for objects that should be synchronized from this node (if primary) to the secondary nodes on startup.

Example:

```
connect-sync /*
```

connect-sync-ignore

Type:	String
Default Value:	N/A
Required:	No

Pattern match for objects that should **not** be synchronized from this node (if primary) to the secondary nodes on startup. There can be multiple instances of this option, one for each pattern of object name that should **not** be synchronized.

Tip: This option should be used in conjunction with the `connect-sync` option to restrict the objects that are synchronized. Typically it is not desirable for `/SYSTEM/*` or `/ALERT/*` to be synchronized, so the examples given here show that configuration.

Example:

```
connect-sync-ignore /SYSTEM/*
connect-sync-ignore /ALERT/*
```

auto-distribute

Type:	String
Default Value:	N/A
Required:	No

Pattern match for objects that should have updates be automatically distributed from this node (if primary) to the secondary nodes during operation. Using this configuration option means that data can be sent in to only one PriceMaster instance, if desired, and automatically distributed to all connected secondary nodes in the cluster.

Tip: It may be desirable to use multiple connections from the input DataSource(s), one to each PriceMaster, rather than this option. However, in the case where only a single data entry point is desired this option can be used.

Example:

```
auto-distribute /*
```

auto-distribute-ignore

Type:	String
Default Value:	N/A
Required:	No

Pattern match for objects that should **not** have updates automatically distributed from this node (if primary) to the secondary nodes during operation. There can be multiple instances of this option, one for each pattern of object name that should **not** have updates automatically distributed.

Tip: This option should be used in conjunction with the auto-distribute option to restrict the objects that are synchronized. Typically it is not desirable for /SYSTEM/* or /ALERT/* to be auto distributed, so the examples given here show that configuration.

Example:

```
auto-distribute-ignore /SYSTEM/*
auto-distribute-ignore /ALERT/*
```

cluster-nosave-on-secondary

Type:	Boolean
Default Value:	FALSE
Required:	No

Activating this option prevents the transformer cache from saving (persisting) replicated objects if it is a secondary node when shut down. This option would typically be used in conjunction with the connect-sync and auto-distribute options.

Tip: This option should be used in conjunction with the auto-distribute option to restrict the objects that are synchronized. Typically it is not desirable for /SYSTEM/* or /ALERT/* to be auto distributed, so the examples given here show that configuration.

Example:

```
cluster-nosave-on-secondary
```

stable-time

Type:	Integer
Default Value:	5
Required:	No

The time, in seconds, after the instance has started up, to wait before checking and making the node primary. This delay prevents a mis-configured system from starting and stopping repeatedly and changing the cluster status unnecessarily.

Example:

```
stable-time 10
```

check-period

Type:	Float
Default Value:	1.0
Required:	No

The time, in seconds, after the instance has started up, to wait before checking and making the node primary. This delay prevents a mis-configured system from starting and stopping repeatedly and changing the cluster status unnecessarily.

Example:

```
check-period 2.0
```

log-file

Type:	String
Default Value:	cluster.log
Required:	No

The file that should be used for logging messages generated by the cluster module.

Example:

```
log-file cluster-pm1.log
```

log-level

Type:	String
Default Value:	INFO
Required:	No

The maximum level of messages that will be logged. This value should be one of ERROR, WARN, NOTIFY, INFO, DEBUG. In normal operation the INFO level should be used to avoid generating excessively large logs. For troubleshooting purposes the DEBUG level should be used and will output the most detailed information.

Example:

```
log-level DEBUG
```

7.4 Page Construction Module

The page construction module is responsible for creating pages from text, records and sections of other pages.

The page construction module is loaded using the `add-module pagecons` configuration option in *transformer.conf*

Configuration Options

The following options can be used to configure the page construction module. These options can be changed in the *etc/pagecons.conf* file and will be read on startup of the PriceMaster.

Tip: The templates that are used to construct pages are created and edited using the Page Constructor tab of the PriceMaster console and stored using XML files in the directory defined by the **template-dir** configuration option.

template-dir

Type:	String
Default Value:	%r/templates
Required:	Yes

The directory that is used to store the runtime configuration XML files. These files define the page templates.

Example:

```
template-dir %r/etc/templates
```

log-file

Type:	String
Default Value:	pagecons.log
Required:	No

The file that should be used for logging messages generated by the page construction module.

Example:

```
log-file pagecons-pm1.log
```

log-level

Type:	String
Default Value:	INFO

Required:	No

The maximum level of messages that will be logged. This value should be one of ERROR, WARN, NOTIFY, INFO, DEBUG. In normal operation the INFO level should be used to avoid generating excessively large logs. For troubleshooting purposes the DEBUG level should be used and will output the most detailed information.

Example:

```
log-level DEBUG
```

7.5 Permissioning Module

The permissioning module is responsible for controlling vendor access to individual objects. It is also responsible for mapping input object and field names to those understood or expected by the vendor.

The permissioning module is loaded using the `add-module pmxauth` configuration option in *transformer.conf*

Note: The permissioning module is a core part of PriceMaster and should always be loaded. It is not advised that the above configuration is removed from *transformer.conf*

Configuration Options

The following options can be used to configure the permissioning module. These options can be changed in the *etc/pmxauth.conf* file and will be read on startup of the PriceMaster.

Tip: Runtime configuration of the vendor permissions and object / field mapping is carried out using the Permissions Editor tab of the PriceMaster console.

perm-dir

Type:	String
Default Value:	%r/perm
Required:	Yes

The directory that is used to store the runtime configuration XML files. These files define the permissions and mappings for each output peer.

Example:

```
perm-dir %r/etc/perm
```

vendorack-enable

Type:	Boolean
Default Value:	FALSE
Required:	No

Enables the vendor acknowledgement behaviour.

Example:

```
vendorack-enable
```

vendorack-seqnum-field

Type:	String
Default Value:	VendorAckSeqnum
Required:	No

If using the vendor acknowledgement behaviour (enabled using the `vendorack-enable` option), this field will be used to return the sequence number of the message being acknowledged.

Example:

```
vendorack-seqnum-field    AckSequenceNumber
```

vendorack-symbol-field

Type:	String
Default Value:	VendorAckFeedSymbolName
Required:	No

If using the vendor acknowledgement behaviour (enabled using the `vendorack-enable` option), this field will be used to return the object name of the message being acknowledged.

Example:

```
vendorack-symbol-field    AckObjectName
```

expand-page-images

Type:	Boolean
Default Value:	FALSE
Required:	No

If enabled then full page images will always be sent, including areas that are blanked, when an output requests the page from the PriceMaster core.

Example:

```
expand-page-images
```

log-file

Type:	String
Default Value:	pmxauth.log
Required:	No

The file that should be used for logging messages generated by the permissioning module.

Example:

```
log-file pmxauth-pm1.log
```

log-level

Type:	String

Default Value:	INFO
Required:	No

The maximum level of messages that will be logged. This value should be one of ERROR, WARN, NOTIFY, INFO, DEBUG. In normal operation the INFO level should be used to avoid generating excessively large logs. For troubleshooting purposes the DEBUG level should be used and will output the most detailed information.

Example:

```
log-level DEBUG
```

7.6 UMDP Output Handler

The UMDP Output Handler (*umdpsink*) is used to send data using the UMDP protocol (MarketLink). One instance of *umdpsink* is required for each vendor connected to PriceMaster receiving UMDP.

> **Tip:** As with all PriceMaster Output Handlers, umdpsink is a Caplin DataSource and is configured to connect to the PriceMaster using the standard add-peer / end-peer configuration group. This section of the document only contains configuration information specific to *umdpsink*. For more details on the standard configuration options common to all DataSource applications please refer to the Transformer Administration Guide, included in the PriceMaster kit.

See Adding an Output for details of how to add a new instance of *umdpsink* to a PriceMaster configuration.

Configuration Options

The following options can be used to configure *umdpsink*. These options can be changed in the *etc/umdpsink[n].conf* file (where [n] is the output number of the *umdpsink* instance). This configuration file will be read on startup of the Output Handler.

umdp-listen-port

Type:	Integer
Default Value:	N/A
Required:	Yes

The TCP port on which *umdpsink* will listen for an incoming vendor connection. This should be unique for each *umdpsink* instance / vendor connection.

Example:

```
umdp-listen-port    10011
```

serial-port

Type:	Integer
Default Value:	N/A
Required:	No

The serial port to use, if the vendor is using a serial port connection.

> **Note:** If this option is used then TCP is disabled and the **umdp-listen-port** configuration option is ignored

Example:

```
serial-port   /dev/ttyb
```

serial-baudrate

Type:	Integer
Default Value:	N/A
Required:	No

The serial baud rate to use, if the vendor is using a serial port connection.

Note: If this option is used then TCP is disabled and the **umdp-listen-port** configuration option is ignored

Example:

```
serial-baudrate    115200
```

vendorack-enable

Type:	Boolean
Default Value:	FALSE
Required:	No

Enables the vendor acknowledgement behaviour.

Example:

```
vendorack-enable
```

vendorack-seqnum-field

Type:	String
Default Value:	VendorAckSeqnum
Required:	No

If using the vendor acknowledgement behaviour (enabled using the `vendorack-enable` option), this field will be used to return the sequence number of the message being acknowledged.

Example:

```
vendorack-seqnum-field    AckSequenceNumber
```

vendorack-symbol-field

Type:	String
Default Value:	VendorAckFeedSymbolName
Required:	No

If using the vendor acknowledgement behaviour (enabled using the `vendorack-enable` option), this field will be used to return the object name of the message being acknowledged.

Example:

```
vendorack-symbol-field    AckObjectName
```

buffer-len

Type:	Integer
Default Value:	2048
Required:	No

Length of receiving buffer.

Example:

```
buffer-len    8192
```

contributor-id

Type:	String
Default Value:	
Required:	No

Contributor identifier to send with UMDP packets (max 7 chars).

Example:

```
contributor-id    CAPLIN2
```

no-acks

Type:	Boolean
Default Value:	FALSE
Required:	No

Whether to wait for ACK messages before sending the next packet (synchronous operation).

Tip: This option should not be used unless the vendor indicates it is required, and never when running with a serial connection.

Example:

```
no-acks
```

heartbeat-period

Type:	Integer
Default Value:	30
Required:	No

The number of seconds between heartbeat packets sent over UMDP.

Example:

```
heartbeat-period    15
```

suppress-page-topline

Type:	Boolean
Default Value:	FALSE
Required:	No

If enabled, the top line of 25x80 pages will not be sent.

Tip: This option may be required if the vendor populates the top line of the page with their own data.

Example:

```
auto-distribute-ignore /SYSTEM/*
auto-distribute-ignore /ALERT/*
```

client-name

Type:	String
Default Value:	N/A
Required:	No

The name of the output vendor, as it should appear in the PriceMaster console.

Example:

```
client-name    Reuters-1
```

log-file

Type:	String
Default Value:	event-umdpsink[n].log
Required:	No

The file that should be used for logging messages generated by *umdpsink*.

Example:

```
log-file    umdpsink0-pm1.log
```

log-level

Type:	String
Default Value:	INFO
Required:	No

The maximum level of messages that will be logged. This value should be one of ERROR, WARN, NOTIFY, INFO, DEBUG. In normal operation the INFO level should be used to avoid generating excessively large logs. For troubleshooting purposes the DEBUG level should be used and will output the most detailed information.

Example:

```
log-level    DEBUG
```

7.7 MPF Output Handler

The MPF Output Handler (*mpfsink*) is used to send data using the MPF protocol (Bloomberg). One instance of *mpfsink* is required for each vendor connected to PriceMaster receiving MPF. Typically only Bloomberg will receive the MPF protocol.

Tip: As with all PriceMaster Output Handlers, *mpfsink* is a Caplin DataSource and is configured to connect to the PriceMaster using the standard add-peer / end-peer configuration group. This section of the document only contains configuration information specific to *mpfsink*. For more details on the standard configuration options common to all DataSource applications please refer to the Transformer Administration Guide, included in the PriceMaster kit.

See Adding an Output for details of how to add a new instance of *mpfsink* to a PriceMaster configuration.

Note: Currently only packet types 2 (records), 5 (heatbeats) and 9 (standard pages) are supported. Future versions of mpfsink may support further packet types including market depth records and attribute pages.

Configuration Options

The following options can be used to configure *mpfsink*. These options can be changed in the *etc/mpfsink[n].conf* file (where [n] is the output number of the *mpfsink* instance). This configuration file will be read on startup of the Output Handler.

mpf-contrib-server

Type:	String
Default Value:	N/A
Required:	Yes

The host name or IP address of the Bloomberg server that will be connected to.

Example:

```
mpf-contrib-server    182.43.12.101
```

mpf-contrib-port

Type:	Integer
Default Value:	N/A
Required:	Yes

The port of the Bloomberg server that will be connected to.

Note: If this option is used then TCP is disabled and the **umdp-listen-port** configuration option is ignored

Example:

```
mpf-contrib-port   18235
```

broker-code

Type:	String
Default Value:	N/A
Required:	Yes

The 4 character string to be used for identifying the contributor organization to Bloomberg.

Example:

```
broker-code    CAPL
```

city-code

Type:	String
Default Value:	N/A
Required:	Yes

The 3 character string to be used for identifying the contributor location to Bloomberg.

Tip: The value for this configuration option (and the **broker-code** option) are usually provided by Bloomberg to uniquely identify a contributor. If you don't have this information then contact Bloomberg.

Example:

```
city-code    LON
```

dsfield-mpftt

Type:	String Array
Default Value:	N/A
Required:	No

This option is used to configure incoming DataSource field to outgoing Bloomberg transaction type code mappings. There should be one instance for each field to be mapped to Bloomberg transaction type code.

Note: The provided configuration should typically not need to be adjusted. The incoming DataSource fields are defined in `fields-bloomberg.conf` which is shared between the Transformer process and the *mpfsink* instances. The DataSource field names (e.g. **bAsk**) should be used in PriceMaster for output fields when configuring field mapping using the PriceMaster Console.

Example:

```
dsfield-mpftt  bAsk                         A
dsfield-mpftt  bBid                         B
dsfield-mpftt  bIndex                       D
dsfield-mpftt  bTrade                       T
```

vendorack-seqnum-field

Type:	String
Default Value:	VendorAckSeqnum
Required:	No

If using the vendor acknowledgement behaviour (enabled using the `vendorack-enable` option), this field will be used to return the sequence number of the message being acknowledged.

Example:

```
vendorack-seqnum-field    AckSequenceNumber
```

vendorack-symbol-field

Type:	String
Default Value:	VendorAckFeedSymbolName
Required:	No

If using the vendor acknowledgement behaviour (enabled using the `vendorack-enable` option), this field will be used to return the object name of the message being acknowledged.

Example:

```
vendorack-symbol-field    AckObjectName
```

log-file

Type:	String
Default Value:	event-mpfsink[n].log
Required:	No

The file that should be used for logging messages generated by *mpfsink*

Example:

```
log-file    mpfsink0-pm1.log
```

log-level

Type:	String
Default Value:	INFO
Required:	No

The maximum level of messages that will be logged. This value should be one of ERROR, WARN, NOTIFY, INFO, DEBUG. In normal operation the INFO level should be used to avoid generating excessively large logs. For troubleshooting purposes the DEBUG level should be used and will output the most detailed information.

Example:

```
log-level    DEBUG
```

CAPLIN

Triton Court
14 Finsbury Square
London EC2A 1BR
UK

Telephone:+44 20 7826 9600
Fax:+44 20 7826 9610

www.caplin.com
info@caplin.com