User's Guide

v 4.2

Table of Contents

1 Introduction ... 1
1.1 Unleash your IT ... 1
1.2 Introducing PARE ... 2
1.3 A bit of a background ... 3
1.4 About PARE ... 3
2 Concepts ... 4
2.1 Data ... 5
2.2 Conditions ... 5
2.3 Actions ... 6
2.4 Functions ... 6
2.5 Supporting Assets ... 7
2.6 Overall schematic ... 8
2.7 User Interface ... 8
3 PARE Studio ... 9
3.1 Menu Bar ... 9
3.1.1 Toolbar – New ... 10
3.1.2 Toolbar – Open ... 10
3.1.3 Toolbar – Save ... 10
3.1.4 Toolbar – Save As ... 10
3.1.5 Toolbar – Close ... 10
3.1.6 Toolbar – Partitions ... 11
3.1.7 Toolbar – Import Configuration ... 11
3.1.8 Toolbar – JMX Remote ... 12
3.1.9 Toolbar – Bundles ... 13
3.1.10 Toolbar – Server Admin ... 14
3.1.11 Toolbar – Settings ... 28
3.2 Using the Tree ... 28
3.3 Expressions ... 29
3.4 Script Support ... 31
3.5 PARE Settings and Password Vault ... 33
3.5.1 Instance and Host Settings ... 33
3.5.2 Host Parameters ... 34
3.5.3 Password Vault ... 35
3.5.4 Server Links ... 35
3.5.5 Security Keys ... 36
3.6 Fields ... 37
3.6.1 Common Characteristics ... 37
3.6.2 User Field ... 38
3.6.3 Data Source Field ... 38
3.6.4 Now (date/time) ... 41
3.6.5 Date Diff ... 41
3.6.6 Computed Date ... 42
3.6.7 Composed Field ... 42
3.6.8 OAuth2Token Field ... 43
3.7 Functions ... 44
3.7.1 Built-in Functions ... 44
3.7.2 Custom Functions ... 48
3.7.3 Statistical Functions ... 49
3.7.4 Contains Function ... 50
3.7.5 Socket Read Function ... 51
3.7.6 Server Socket Read Function ... 51
3.7.7 Result Function ... 52
3.7.8 Formatter Functions ... 52
3.7.9 Model Json / XML Functions ... 53
3.7.10 Rest Service functions ... 53

3.7.11 Script Function ... 56
3.7.12 BAPI Function ... 56
3.7.13 SOAP Function ... 57
3.7.14 Case Function ... 59
3.8 Conditions ... 60
3.8.1 Simple Condition ... 60
3.8.2 Expression Condition ... 61
3.8.3 Joined Condition ... 62
3.9 Actions ... 62
3.9.1 Common characteristics of Actions ... 63
3.9.2 Built-in Actions ... 63
3.9.3 Custom Action ... 66
3.9.4 Field Action ... 67
3.9.5 DML Action ... 68
3.9.6 Prepared DML Action ... 68
3.9.7 Stats Refresh Action ... 70
3.9.8 Socket Write Action ... 70
3.9.9 Server Socket Write Action ... 70
3.9.10 TCP Server Start Action ... 71
3.9.11 TCP Server Stop Action ... 72
3.9.12 Rule ... 72
3.9.13 Sequence ... 73
3.9.14 Transaction Action ... 74
3.9.15 Job ... 75
3.9.16 Loop Action ... 77
3.9.17 Field Setter Action ... 78
3.9.18 Decision Table ... 79
3.9.19 Case Table Action ... 80
3.9.20 On Error Action ... 81
3.9.21 Scheduled Action ... 82
3.9.22 Script Actions ... 83
3.9.23 Email Send Action ... 84
3.9.24 Parallel Tasks Action ... 85
3.9.25 Event Action ... 85
3.9.26 Action Graphs ... 87
3.10 Connectors ... 88
3.10.1 DB Connector ... 88
3.10.2 DB Connector Link ... 90
3.10.3 SMTP Connector ... 90
3.10.4 POP3 / IMAP Connector ... 92
3.10.5 FTP Connector ... 92
3.10.6 Class Connector ... 93
3.10.7 Socket connector ... 96
3.10.8 Server Socket Connector ... 96
3.10.9 SAP Connector ... 97
3.10.10 Mongo DB Connector ... 97
3.10.11 SOAP Connector ... 98
3.11 Data Sources ... 98
3.11.1 Object Data Source ... 98
3.11.2 DB Query ... 99
3.11.3 File Data ... 102
3.11.4 Master Data ... 104
3.11.5 XML Data Source ... 104
3.11.6 JSON Data Source ... 106
3.11.7 Email Data Source ... 107
3.11.8 User Data Source ... 108
3.11.9 Excel Data Source ... 109

3.11.10 Mongo DB Data Source ... 110
3.11.11 Filtered Data Source ... 111
3.11.12 Custom Data Source ... 111
3.11.13 Memory Data Source ... 113
3.11.14 Transformer Data Source ... 114
3.12 Models ... 115
3.12.1 Model ... 115
3.12.2 JAR content ... 116
3.13 Plugins ... 117
3.13.1 Plugin Installation ... 117
3.14 Linking another PARE configuration ... 117
3.15 Node List ... 119
3.16 Node Properties ... 119
4 Integrating with PARE ... 120
4.1 PARE Java API integration ... 120
4.2 Using a HTTP Client (OkHttp) to access PARE REST API ... 121
4.3 Legacy asset integration via Class Connector ... 122
4.4 Existing Web Service integration via PARE SOAP and HTTP Clients ... 122
5 PARE Server ... 122
5.1 Overall Architecture ... 123
5.2 Core concepts ... 123
5.2.1 Engines ... 123
5.2.2 Partitions ... 123
5.2.3 Access Control ... 124
5.2.4 Access Tokens ... 124
5.3 PARE Server HTTP API ... 125
5.3.1 Hardening PARE Server Security ... 125
6 Optimizing for performance ... 126
7 PARE Workflows (BPM Engine) ... 127
7.1 Concepts ... 127
7.1.1 Task States ... 127
7.1.2 Relationships (Triggers) ... 127
7.1.3 Task Types ... 128
7.1.4 Workflow Data ... 129
7.1.5 Conditional Branching ... 130
7.2 Workflow Editor ... 131
7.2.1 Manual Task ... 131
7.2.2 Automatic Task ... 133
7.2.3 Waiter Task ... 134
7.2.4 Flow Task ... 135
7.2.5 Workflow Visualizer ... 136
7.3 Workflow Monitor ... 136
7.3.1 Workflow Instance data retention period ... 137
7.4 Workflow Built-ins ... 138
7.5 Workflow API ... 138
8 Document Management System (PARE DMS) ... 139
8.1 DMS feature highlights ... 139
8.1.1 Attribute based search ... 139
8.1.2 Access Control ... 140
8.1.3 Common or Engine level storage ... 140
8.1.4 Automatic detection of Content Type ... 140
8.1.5 DMS API ... 140
8.1.6 DMS Built-ins ... 140
8.1.7 Studio GUI ... 141
9 Licensing ... 142

Table of Figures

Figure 1 – PARE Engine Components 4
Figure 2 - Field Types 5
Figure 3 - Condition types 5
Figure 4 - Action categories 6
Figure 5 - Function categories 7
Figure 6 - Supporting Assets 7
Figure 7 – Overall Schematic 8
Figure 8 – Main Window 9
Figure 9 – Main menu 9
Figure 10 – Toolbar 9
Figure 11 - File Open dialog 10
Figure 12 - Partition manager 11
Figure 13 - Import Node Selection dialog 12
Figure 14 - JMX Controller 13
Figure 15 - Toolbar Bundles 13
Figure 16 - PARE Server administration 14
Figure 17 - User manager 15
Figure 18 – User Properties Dialog 15
Figure 19 - User Security Questions dialog 16
Figure 20 - User Attributes dialog 17
Figure 21 - Availability configuration 17
Figure 22 - Manage Workgroups 18
Figure 23 - Set engine permissions 18
Figure 24 - Engine manager 19
Figure 25 - Creating a Webhook 20
Figure 26 - Open Create Asset Link dialog 21
Figure 27 – Asset properties dialog 21
Figure 28 - Rate Limiter Configuration 22
Figure 29 - Manage remote parameters 22
Figure 30 - Remote Partition Manager 23
Figure 31 – Joins the remote Server to a Cluster 23
Figure 32 - PARE cluster overview 24
Figure 33 - Remote Files manager 24
Figure 34 - Remote Auth sources 25
Figure 35 - Auth IDP configuration 26
Figure 36 - Remote Settings panel 27
Figure 37 - Configure Holiday Calendars 28
Figure 38 – Root menu 28
Figure 39 – Node Category menu 29
Figure 40 – Node menu 29
Figure 41 - Expression Helper popup 31
Figure 42 - Script execution flow 32
Figure 43 - Script popup menu 32
Figure 44 - Instance and Host Settings 33
Figure 45 - Settings editor 34
Figure 46 - Expression popup 34
Figure 47 - Password store editor 35
Figure 48 - Using the Password Manager 35
Figure 49 - Server Link editor 35
Figure 50 - Security key setter 36
Figure 51 – Add Fields menu 37
Figure 52 – User Field editor 38
Figure 53 – Data Source Field editor 38
Figure 54 – Data Source Field editor 39

Figure 55 – RDBMS Data Source Field schematic 39
Figure 56 – File Data Source Field schematic 39
Figure 57 – Object Field schematic 40
Figure 58 - Create Datasource Fields 40
Figure 59 - Select Datasource Columns 40
Figure 60 – Now Field editor 41
Figure 61 – Date Diff Field editor 41
Figure 62 – Computed Date editor 42
Figure 63 – Composed Field editor 42
Figure 64 - Oauth2 Token field editor 43
Figure 65 – Add Function menu 44
Figure 66 – Built-in Function viewer 44
Figure 67 – Recommended Eclipse Java compiler options 49
Figure 68 – Stats Function 49
Figure 69 – Create Stats Functions 50
Figure 70 - Select Stats Functions to create 50
Figure 71 - Create Contains function 50
Figure 72 - Create Socket Read Function 51
Figure 73 - Create TCP Server Socket Read function 51
Figure 74 - Result Function editor 52
Figure 75 - Create Formatter Function 53
Figure 76 - Create Model JSON / XML Functions 53
Figure 77 - Rest Service function editor 54
Figure 78 - Authorization options 54
Figure 79 – HTTP Header options 54
Figure 80 - Body content - form type 55
Figure 81 - Body content - raw type 55
Figure 82 - Proxy configuration 55
Figure 83 - Script Function editor 56
Figure 84 - BAPI Method selector 57
Figure 85 - BAPI Methods in tree browser 57
Figure 86 - Soap Function editor 58
Figure 87 - SOAP Operation chooser 58
Figure 88 - Care Function Panel 59
Figure 89 – Add Condition menu 60
Figure 90 – Simple Condition editor 60
Figure 91 – Expression Condition editor 61
Figure 92 – Joined Condition editor 62
Figure 93 – Add Action menu 63
Figure 94 – Custom Action editor 66
Figure 95 – Field Action editor 67
Figure 96 – DML Action editor 68
Figure 97 – DML Creation Menu 68
Figure 98 - Prepared DML Action editor 69
Figure 99 – Stats Refresh Action Editor 70
Figure 100 - Create Socket Write Action 70
Figure 101 - Create Server Socket Write Action 70
Figure 102 - Create TCP Server Start Action 71
Figure 103 - TCP Server editor 71
Figure 104 - Create TCP Server Stop Action 72
Figure 105 – Rule editor 72
Figure 106 – Sequence editor 73
Figure 107 - Transaction Action editor 74
Figure 108 – Job editor 75
Figure 109 - Multi threaded Job processing 76
Figure 110 – Number of threads / times 77
Figure 111 - Loop Action editor 78

Figure 112 – Decision Table editor 79
Figure 113 - Case Table Editor 80
Figure 114 – On Error Action editor 81
Figure 115 - On-Error Action flow 82
Figure 116 - Scheduled Action editor 82
Figure 117 - Script Action editor 83
Figure 118 – Email Send Action editor 84
Figure 119 - Event Action editor 86
Figure 120 - Access the Process Graph 88
Figure 121 - Action Graph viewer 88
Figure 122 – Add Connector menu 88
Figure 123 – DB Connector editor 89
Figure 124 - DB Connector Link editor 90
Figure 125 - SMTP Connector editor 91
Figure 126 - POP3 / IMAP Connection editor 92
Figure 127 - FTP Connector editor 93
Figure 128 - Class Connector editor 94
Figure 129 - Map Object assets 95
Figure 130 - Select Object Assets 95
Figure 131 - Socket Connector editor 96
Figure 132 - Server Socket editor 96
Figure 133 - SAP Connector editor 97
Figure 134 - Mongo DB Connection editor 97
Figure 135 - Soap Connector editor 98
Figure 136 – Data Sources tree 98
Figure 137 – Add Object Data Source 99
Figure 138 – Object Data Source editor 99
Figure 139 – Query Builder 100
Figure 140 – Query String editor 101
Figure 141 – Delimited File editor 102
Figure 142 – Fixed Length File editor 103
Figure 143 – Master Data editor 104
Figure 144 - XML Data Source editor 105
Figure 145 - XML Node browser 105
Figure 146 - JSON Data Source editor 106
Figure 147 - JSON Node Selection editor 106
Figure 148 - Email Data Source editor 107
Figure 149 - User Data Source editor 108
Figure 150 - Excel sample data 109
Figure 151 - Excel Data Source editor 109
Figure 152 - Mongo DB Data source editor 110
Figure 153 - Filtered Data-source editor 111
Figure 154 - Custom DataSource editor 113
Figure 155 - Memory Datasource editor 113
Figure 156 - Transformer Data source editor 114
Figure 157 - Model editor 115
Figure 158 - Model Assets 116
Figure 159 – Plugin View 117
Figure 160 – Configure Link to external configuration 118
Figure 161 - Creation of Linked Configuration 118
Figure 162 – Node list viewer 119
Figure 163 – Properties editor 120
Figure 164 - PARE Server Architecture 123
Figure 165 - Access control structure 124
Figure 166 - Swagger Documentation 125
Figure 167 - Workflow Visualization (the RED line indicates Critical Path) 128
Figure 168 - Workflow Data Template 129

Figure 169 - Inter workflow data transfer mapping 130
Figure 170 - All paths cancelled 130
Figure 171 - Alternative Path chosen 130
Figure 172 - Interim branch cancelled 131
Figure 173 - Workflow editor 131
Figure 174 - Manual Task Editable Fields 132
Figure 175 - Manual Task Attributes 132
Figure 176 - Manual Task Skills 132
Figure 177 - Manual Task Finish Checks 133
Figure 178 - Manual Task Settings 133
Figure 179 - Automatic Task configuration panel 134
Figure 180 - Waiter Task configuration panel 134
Figure 181 - Waiter Task visualization 135
Figure 182 - Flow Task Data Mapping 135
Figure 183 - Flow Task Finish Checks 135
Figure 184 - Flow Task Settings 135
Figure 185 - Workflow Monitor UI Module 136
Figure 186 - Workflow Instance Visualizer 137
Figure 187 – Workflow Instance data editor 137
Figure 188 - Workflow Common API Swagger documentation 138
Figure 189 - Workflow Template Specific API 139
Figure 190 - Files API Swagger Documentation 140
Figure 191 - Files list view 141
Figure 192 - Set File Attributes 141
Figure 193 - List Filter 142
Figure 194 - License Management 142

1 Introduction

1.1 Unleash your IT

Are you saddled with un-wieldy applications that are difficult to tune to your ever-changing business needs? Do you lose your sleep over stability of the myriad integrations among legacy apps? Do you only dream of somehow reducing cost of change? Can you easily move to cloud if required?

You are not alone. It's a shared concern amongst all organizations, who are trying to be at the cutting edge or their line of business and needs agile and responsive IT to support it.

What if there was a method that allowed your team to just address the core problem (say, a change in a business service), and the rest of the stuff can be handled by a COTS platform, without coding? It will reduce turn-around-time by half, reduce testing and QA time by half (most of the bugs seem to crop up from the integration area), and reduce risks. Hence, you will be able to push fixes or new features into production much faster and at lower costs.

But do such systems exist?

Middleware solutions exist in the market that enables data and API integration between applications. The obvious advantage is standardization of connectivity and protocol. It also addresses the P2P connectivity headache by inserting itself as a layer between all apps, and hence can provide a central view and control of all connections among all participating apps. A few of them allow designing of new flows by combining the interfaces (APIs), thus providing orchestration facility.

On the flip side however, the middleware implementation itself is not exactly plug-n-play. All apps must go through some amount of transformation to be able to successfully integrate with the middleware. Customers also need to buy new hardware to run these large systems (middleware) efficiently and avoid losing out on performance.

So, effectively, to solve a problem, new problems are created!

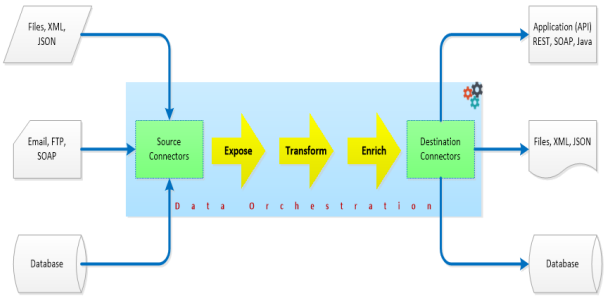
But, imagine there was a COTS platform that can be configured to serve as API gateway for your legacy apps (make it handle all data integration and transformation). Same platform can also consolidate these interfaces (API) and provide central control and manageability just like a middleware, and can orchestrate the APIs, so that new composite services can be created. In short, a software platform that addresses both micro and macro integration requirements and make all the apps in the landscape talk to each other reliably, speedily and often without any changes.

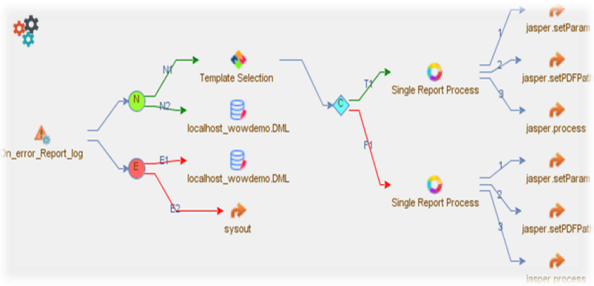
And let the team just concentrate on the business...

1.2 Introducing PARE

(a new approach to App Integration and Orchestration)

Make sense of your data	Give new lease of life to your legacy apps	Extend your IT with new composite services	In short – make your IT *agile*
• Connect with all your data sources • Interpret your data and configure decision making • Enrich, transform, repackage and move your data among apps	• Integrate your legacy apps and give them a new lease of life • Reduce vendor dependency • Avoid costly rip-n-replace projects	• Bring agility to your Customers by easily building composite services by exposing existing assets as API • Enable your Business to respond faster by modelling new processes and instant deployment	• Enable business to respond much faster to changing environment • Reduce time to conceptualize, implement, test and deploy changes in your IT • Save costs

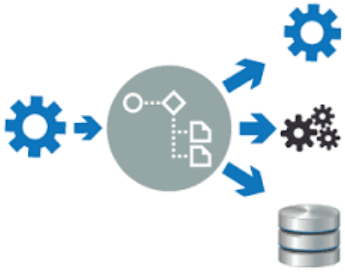


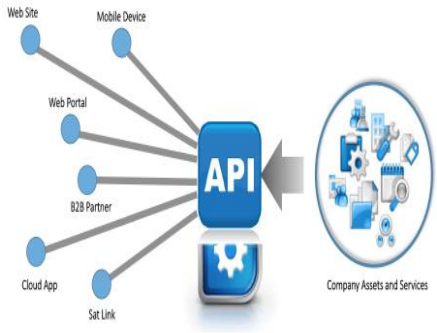


Some common use cases…

Service Orchestration

- Design new processes using various Containers and exposed services
- Inject data into processes and configure decision making
- Package the new process as a service and offer to clients as secure RESTful API
- Zero coding – from concept to deployment



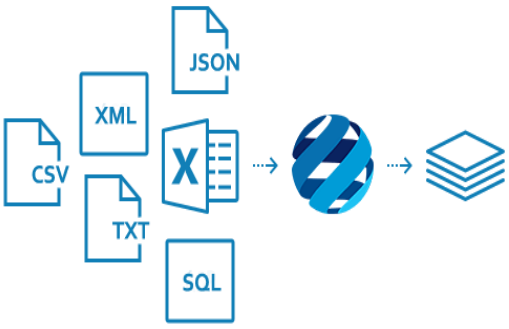


API Gateway (aka. Micro Services for your legacy apps)

- Get insulated from app partitioning or location
- Get optimal API for each client
- Reduce the number of requests/roundtrips
- Simplify the client by moving logic for calling multiple services from the client to API gateway
- Translates from whatever protocols are used internally to a standardized web-friendly API protocol

Data Integration

- Expose data from different systems in your IT landscape
- Enrich, Transform, re-package
- Push to one or more target endpoints
- Build logic to do conditional data extract and load
- Bring agility in your integration architecture



1.3 A bit of a background

Application orchestration is the process of integrating more than one application and/or services together to design and automate composite processes, or synchronize data in real-time. It provides an approach to integration that decouples applications from each other, using standardized method for Data Transformation, Orchestration, Security and Reliability, without degrading Performance.

- ✓ **Data Transformation** (ETL) includes capability to Extract data from different formats (like Database, CSV, web service and so on), Transform them (by changing, enriching, aggregating or segregating), and Load (deliver) them to various end points (like Database, File, Web Service and so on).

- ✓ **Orchestration** includes capability to link various existing IT assets (data and services), within or outside of the organization's boundary, to create 'composite' IT assets. This is used to design 'course-grained' services as required by the business, and it promotes reuse and manageability of the underlying existing IT assets.

- ✓ **Security and Reliability** requirements include capability to enforce secure access and operations on enterprise IT assets (Data and Services) that are exposed to the Orchestration system. Structured error handling capability and activity audit trails should ensure fault traceability, localization and hence reliability of the Orchestrated system.

- ✓ An Orchestration system provides standardized methods to configure connectivity between one of more 3rd party systems. However, in doing so, it should not degrade **Performance** of either system or the sum of the systems combined. The performance benefits accrued through automation of connectivity among the participating systems should more than compensate for the internal overheads of the Orchestration system.

1.4 About PARE

PARE provides a unified integration platform that can create network of Data and API across applications both within and outside of the boundaries of organization's IT landscape.

- ✓ **Versatile and extensible connectivity**
  PARE provides built-in connectors for standard Databases, Email servers, Files (record files, CSV and Excel), and web services and Application Classes. Connectivity can be enhanced by integrating Custom data sources built on standard Java interfaces.

- ✓ **Extensive Data Transformation capability**
  PARE Data system (Fields), provides a powerful yet flexible way of exposing data from various sources in a uniform way. It's easy, for example to match a Field from a Database with another from a File, and update a third inside of an Application, all via configuration.

- ✓ **Versatile and extensible Orchestration capability**
  Using Container Functions and Actions PARE allows designing new composite processes that integrate IT assets from across the organization and outside, easily and reliably. Using Configuration Linking capability and PARE server, these composite services can be linked to even higher-level services thus ensuring modularity and required granularity of the services as required by the business. PARE scripting (using JavaScript and LUA) and direct Java API integration facility can be used to build new services from scratch as well.

- ✓ **Security and Reliability**
  PARE Server implements granular Access Control where the administrator can control CRUD access between a USER and a hosted PARE configuration. PARE Server's access control system allows Permission hierarchy and inheritance.

PARE's structured error-handling capability allows the process designer to handle errors at the right level and automate corrective actions. These include selecting of alternative paths in case of error, capturing Audit information, stop or continue further processing, patching of Data under error scenarios and so on, to give users granular control over error scenarios.

- ✓ **Better Performance of Orchestrated processes**
  PARE employs multiple strategies to improve performance, including:
  - Connector pooling and per thread active connectors for immediate response to on-demand requests
  - Powerful multi-threaded data source iterator for batch loads

- ✓ **PARE as a Rule Engine**
  Rule Engine is a system that enables application developers (organizations' IT department) to decouple business rules from application source code. By doing so, they enable the business users (people with direct knowledge of the business and hence its rules) to directly define, test, deploy and maintain the business rules themselves with little or no intervention by the IT department.

  This is beneficial to both the IT department and the business due to following reasons:
  - Business rules are linked to business environment which are highly dynamic. Hence flexibility in defining and changing the rules (basically managing how the system behaves) without IT intervention saves time and enables the business to react faster to a given situation.
  - Keeping the business conditions and resulting actions (rules) outside of main application code flow enables much smaller main code base and a much cleaner and maintainable application. It also enables much faster turnaround time w.r.t change management.
  - Version control of the business rules is separated from main application, and hence, same application code may support various similar business scenarios just by implementing different versions of the Rule Configuration.

  In short, a Rule Engine helps separate the knowledge aspects of the business from the technical aspects of the application, and allows Business Users to directly manipulate the knowledge aspects of the application.

  Combining PARE's extensible Data processing capabilities and App integration, users can easily design Business Rules and integrate with existing Apps.

2 Concepts

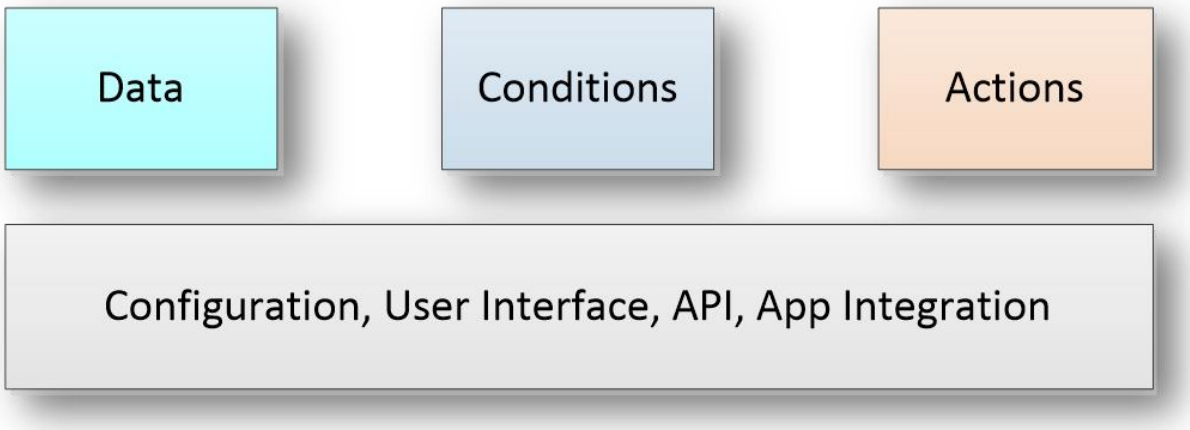


Figure 1 – PARE Engine Components

The main components of PARE are:

2.1 Data

Data drives businesses, and hence PARE puts a lot on emphasis in exposing Data.

Figure 2 - Field Types

User Fields
User defined values are stored. These can be used to define constant values, or to act as temporary data-holders during processing. All user fields support both 'get' and 'set' operations.

Data Source Fields
Value of these fields depend on an underlying data source which can be a database, a file (of various formats), an email server, application data, or custom developed. Some of the underlying data sources allows both 'get' and 'set' operations on the value of the fields. For example, Data Source fields can be automatically created from a SQL Query on any of the supported databases.

Computed Fields
Computed Fields can mix other fields (and Functions) to return a composite value. For example, it's possible to create a computed field that fires a web service to retrieve remote data every-time it's accessed. Another example is creating a computed field that auto-calculates the age of a person by subtracting the DOB (a Data Source Field) from Current Date (a function), in YEARS. Computed Fields are read-only – i.e. they do not support 'set' operations.

Details of each PARE field type can be found here.

2.2 Conditions

Conditions are required to make business decisions which can be automated in PARE process design. Conditions can be used to define 2-way branched processing, or multiple conditions and actions can be tied together to create a Decision Table. PARE allows configuration of complex conditions easily.

Figure 3 - Condition types

Field Conditions (also called Simple Condition)
This is standard comparison between any two fields. Users can compare if Field 1 is greater that, or less than, or equal to Field 2. If Field 2 is backed by a Data Source, additional comparators like IN and NOT_IN are allowed. While Condition Expressions can provide similar functionality also, currently Field Conditions are required when comparing 2 DATE/TIME fields.

Condition Expressions
PARE includes a Condition expression parser that can parse a String with comparison operators and returns a boolean value (true or false). Supported comparison operators include: >, <, >=,

<=, ==, !=, &&, || and brackets (). It's possible to write condition expressions like: ([field1] > [field2] && [field3] == [field4]) || ([field5] != [field6]).

Joined Conditions
Joined Conditions can join one or more conditions in 'AND' or 'OR' condition bundle. Joined conditions can also include other Joined Conditions in its bundle, effectively allowing user to configure conditions to any level of complexity.

Details of each PARE condition type can be found here.

2.3 Actions

Actions are the work done by the Business in response to a certain condition. PARE supports both conditional and non-conditional firing of an Action. Following are the different Action categories:

Figure 4 - Action categories

Custom Actions
Custom Actions are user defined – in Java. Users can create Java code for doing specific things, which can be exposed to PARE via a Class Connector. This feature can be used to expose existing IT assets (in form of Java API) inside PARE. For example, if the user already has API for pushing a message into an existing middleware, the same can be integrated in PARE using Custom Action. Custom Actions, like any other Action, accepts parameters and the parameter types must match with any of the PARE Data Types.

Point Feature Actions
PARE provides specific commonly used capabilities as Point Feature Actions. These include: DML Actions (for calling INSERT, UPDATE, DELETE, TRUNCATE on an RDBMS), Send Email Action (for sending emails via a SMTP Connector), Field Actions (for setting Field values), DataSource Actions (for forward movement and rewind of a Data Source) among others.

Container Actions
The real power of PARE Orchestration is implemented using Container Actions. These Actions basically call other Actions and users can attach various conditions and resources to these containers to control the process flow. Examples of some Container Actions are: Rules (two-way branching based on a Condition), Jobs (iterates a DataSource and calls the set of contained Actions per row of the DataSource), On-Error Actions (traps errors, and does on-error processing), Scheduled Actions (for scheduling contained Actions using internal scheduler), Decision Tables (connects multiple Conditions and Actions together), Sequences (calls a set of contained Actions in sequence), Script Actions (executes a Script, where user can write the flow just like normal coding) among others.

Details of each PARE Action type can be found here.

2.4 Functions

Functions does some things and return a value, and hence can be embedded within PARE Expressions. They are architecturally very similar to Actions. The basic difference between a Function and an Actions are as follows:

- A Function returns a value, whereas an Action does not
- All Actions can be time-limited (start and end validity dates), which is not applicable to Functions

Following are the broad categories of Functions:

Figure 5 - Function categories

User Functions
User functions are conceptually same as Custom Actions. User's Java methods can be exposed as User Functions in PARE via a Class Connector. Regarding the example provided in the description of Custom Actions, if the user already has API for reading a message from an existing middleware, the same can be integrated in PARE using Custom Function. Custom Functions, like any other Function, accepts parameters and the parameter types must match with any of the PARE Data Types.

Point Feature Functions
PARE provides specific commonly used capabilities as Point Feature Functions. These include: Rest Service Functions (a RESTful Web Service caller), Stats Functions (applicable to Data Source Fields) among others.

Container Functions
Container Functions can contain child Actions that can be executed before returning the result of the Function. PARE provides two types of Container Functions: Result Functions (executes the set of contained Actions and returns the value of a configured PARE expression), Model JSON Functions (executes the set of contained Actions and returns the JSON string for the corresponding PARE Model), Script Functions (executes a Script, where user can write the flow just like normal coding, and return a value) among others.

Details of each PARE Function type can be found here.

2.5 Supporting Assets

PARE process design is done using DATA, CONDITIONS, FUNCTIONS and ACTIONS. However, there are supporting resources that help in execution of the Functions or Actions, providing Data to the Fields and so on. These resources (assets) can be categorized as follows:

Figure 6 - Supporting Assets

Data Sources
These are Data Providers for Data Source Fields. Types of Data Sources include: DB Query (contains a SQL Query, that is executed on demand), Record Files (fixed length or delimited), Excel (MS Excel file), Custom (user provided Data Source that implements the TabularData interface), Filtered (a wrapper data source that applies filtration criteria on an already configured Data Source), Email (exposes an email folder via the POP3 or IMAP protocol) among others. Refer here and here for more details.

Connectors
Connectors connect to a specific resource and handles the communication protocol. Examples of Connectors include: DB Connector (provides JDBC connection to a RDBMS), SMTP Connector (provides SMTP session to an Email Send Action), POP3 / IMAP Connector (provides read

capability to the Email Data Source), FTP Connector (provides FTP session to the FTP Upload Action, FTP Download Action and FTP List Function) and Class Connector (loads a Class and optionally instantiates Objects for use by Object Data Source, Custom Function and Custom Actions). Refer here for more details.

Built-in Resources
PARE provides a set of Built-in Functions and Actions inside the Built-in plugin. These are commonly used, and accessible to all PARE instances automatically. There include common Functions like 'substring' (for cutting a String) or 'timestamp' (returns the current time-stamp) among many others and Actions like 'sysout' (print a line in console) or 'command' (executes a OS command) among others. Refer here and here for more details.

2.6 Overall schematic

An overall schematic is provided below, before delving deeper into individual entities:

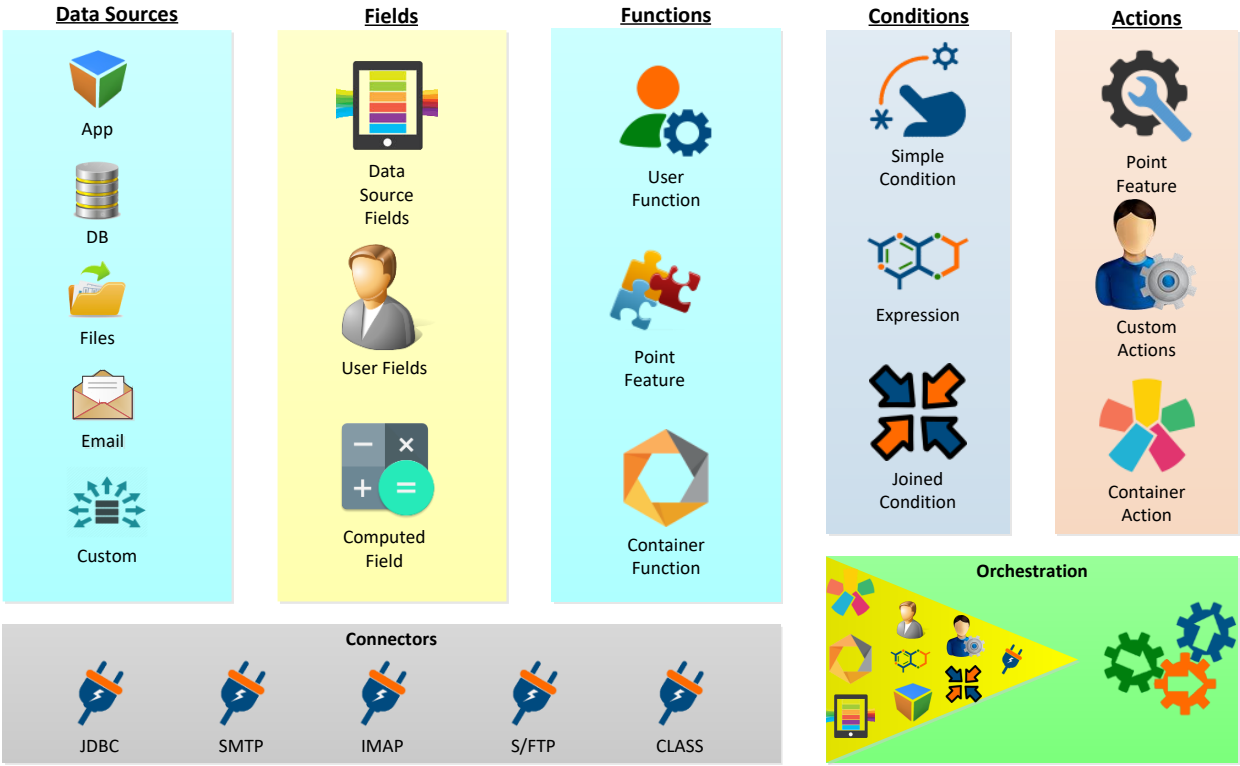


Figure 7 – Overall Schematic

The rest of the User's Guide is divided into 2 sections:

- PARE Studio GUI
- PARE API

2.7 User Interface

PARE Studio is makes it simple to configure PARE and the engine API makes it trivial to integrate it in your IT landscape.

3 PARE Studio

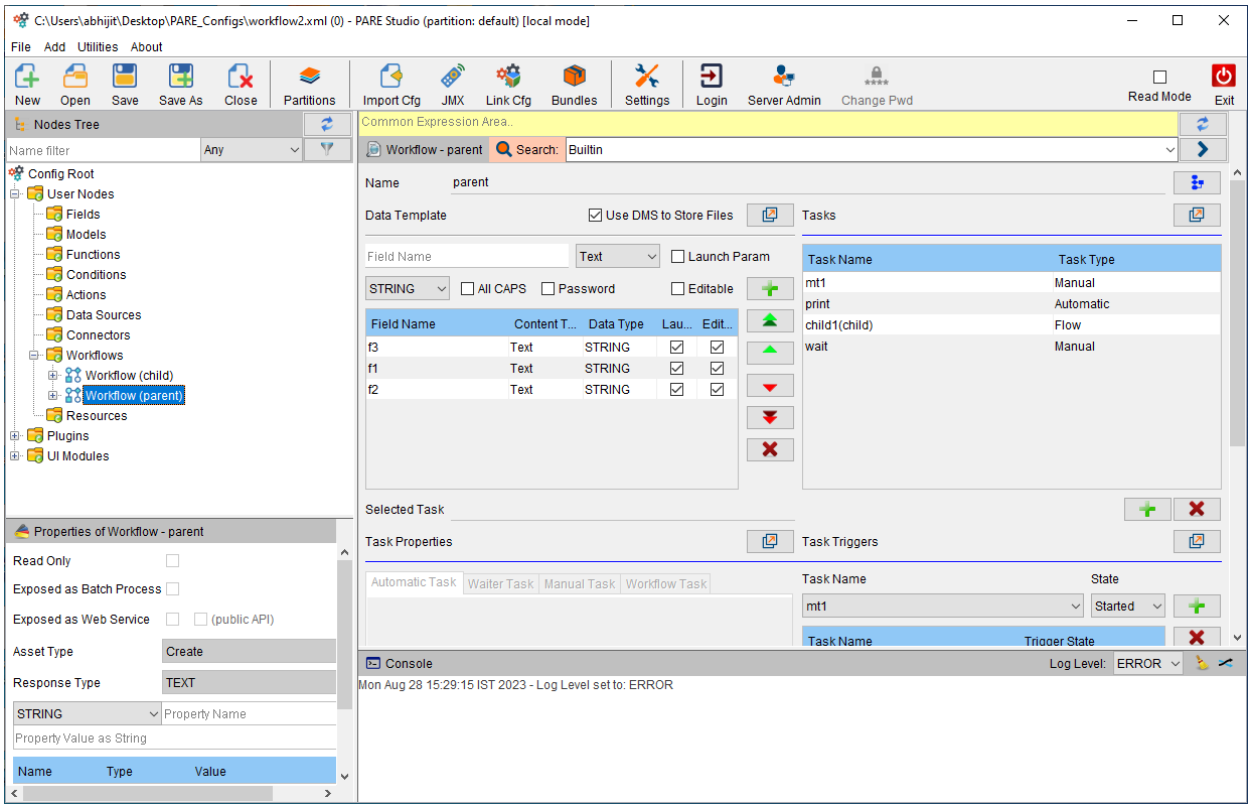


Figure 8 – Main Window

The Studio is used to manage the entire Configuration. It includes all methods to create Fields, Conditions and Actions. It contains a Menu bar, a tool bar, a left tree navigation pane and the main editor area.

The main purpose of the Studio is to produce a XML file with all Configurations. The same can be used by end applications using the Engine API or PARE Server.

3.1 Menu Bar

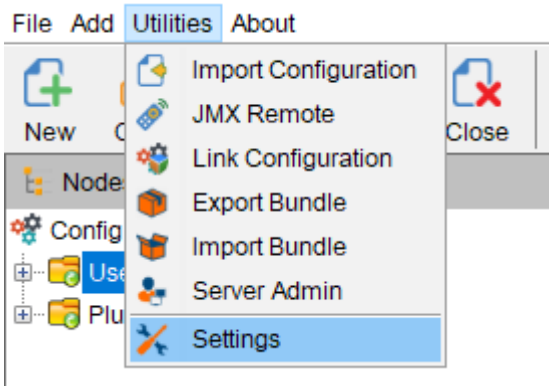


Figure 9 – Main menu

The Menu Bar includes standard items for open, close, save, save-as operations. The same functionality can be accesses via the tool bar buttons also.

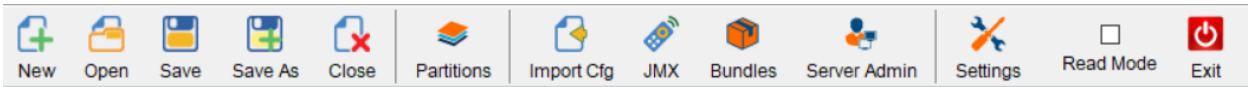


Figure 10 – Toolbar

In addition, the Menu Bar contains a dynamic 'Recent Documents' menu, that remembers few of the recently used documents that were loaded in the Studio.

The functions of the button are self-explanatory. Few things to note are:

- Default file extension is .XML
- The File open dialog always opens the last used directory by default
- If a file from Recent Documents fails to load, the same is removed from that list
- Selecting Open option automatically calls Close for the current document, and if there is unsaved content, user is prompted accordingly. User has option to save current document first, leave it unsaved or cancel the open operation.

3.1.1 Toolbar – New

Cleans up any user nodes and cleans up the slate. In case there are unsaved changes at the time this button is clicked, user is prompted with a suitable warning message and option to save.

3.1.2 Toolbar – Open

This button will first try to cleanup any existing configuration – hence a suitable warning will be shown if there are unsaved changes. Once cleared, it will open the File Open dialog with XML extension filter applied.

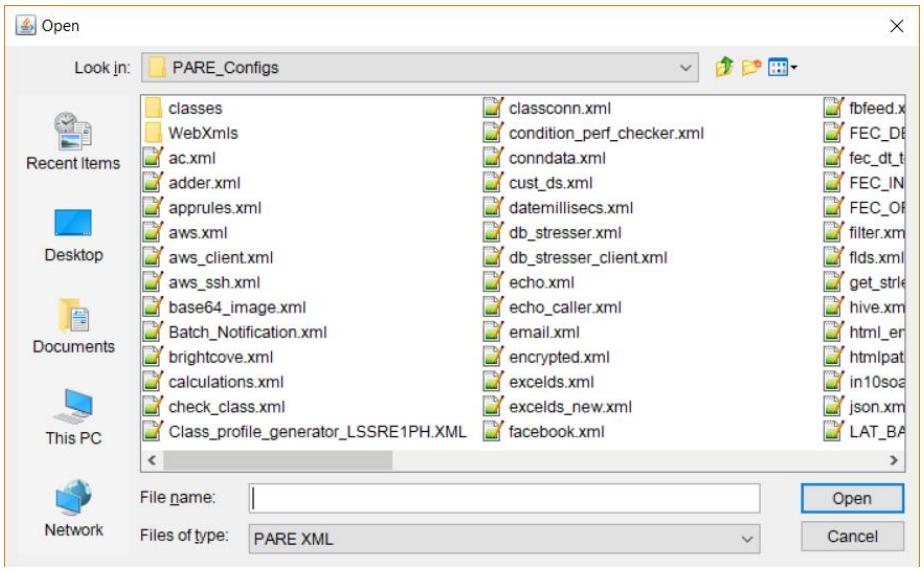


Figure 11 - File Open dialog

3.1.3 Toolbar – Save

For New configurations, this will display the File Save dialog with XML extension filter and allows user to select a filename. Once a filename is supplied (either through a Save, Save As or Open operation), subsequent clicks on this button perform a silent save operation with the same filename.

3.1.4 Toolbar – Save As

Always pops up the File Save dialog with XML extension filter applied. This is required when user needs to save an existing configuration in a different name.

3.1.5 Toolbar – Close

Close button works the same way as New – in-fact the same underlying code-path is executed.

3.1.6 Toolbar – Partitions

This button opens the Partition Selector.

3.1.6.1 Concept of a Partition

A partition defines an 'environment' in which PARE is operating. This includes PARE Settings, PARE Passwords and PARE Local Parameters. Using partitions, it's possible to operate PARE under different environments using the same JVM. Some sample use cases of this feature are:

- Define Test or Production environments
- Implement multi-tenancy in a cloud environment

For example, if the DB connection string (defined in the DB Connector) is defined in PARE Local Parameters (and the DB password is defined in PARE Passwords), then using PARE Partitions, its possible to connect to different Databases using the same PARE Configuration XML. Just by selecting a different Partition in the Studio, it's possible to easily switch environments.

PARE Server extends the concept of partitions to 'users'. A user can belong to only ONE partition. When an API request comes from a user, the corresponding partition is automatically activated before invoking the API. Hence, multi-tenancy can be implemented easily by defining a partition per tenant and defining an existing partition per user. Underlying technology ensures that same API endpoint is invoked under the calling user's partition context.

3.1.6.2 Partition Manager

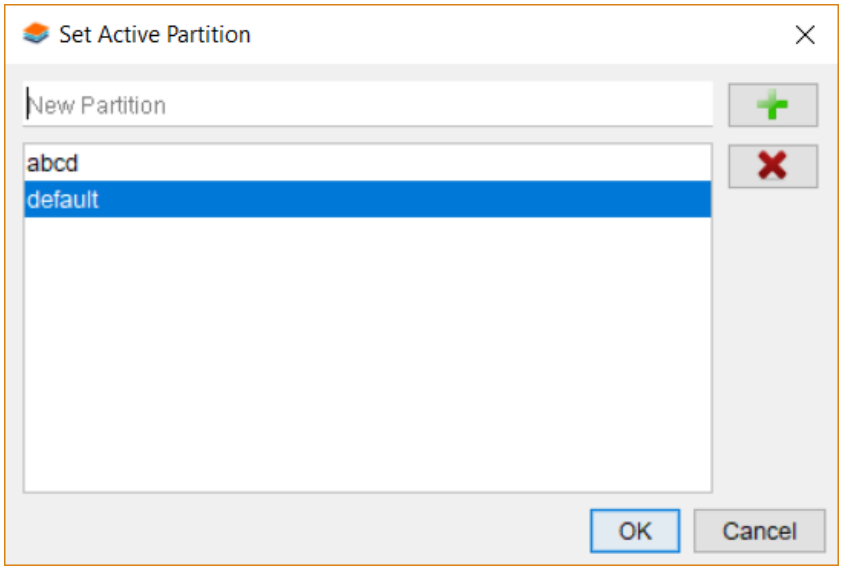


Figure 12 - Partition manager

The Partition manager allows creation of new partitions and selection of the currently active partition. The default partition name is 'default'. This is created automatically when PARE is first used and is present in all PARE instances. This is a reserved partition and cannot be deleted.

New partitions can be created and existing partitions (except 'default') can always be deleted. The list view always highlights the currently active partition. PARE Studio allows only ONE Active partition at any given point. PARE Server however does not have such restrictions and can serve API from the context of any of the existing partitions simultaneously.

3.1.7 Toolbar – Import Configuration

The Import button enables importing another configuration into the existing one. This is one of the ways in which existing PARE Assets can be re-used. For example, a DB connection object can be imported from an existing XML. Import feature allows user to select a sub-set of nodes to be imported from target configuration:

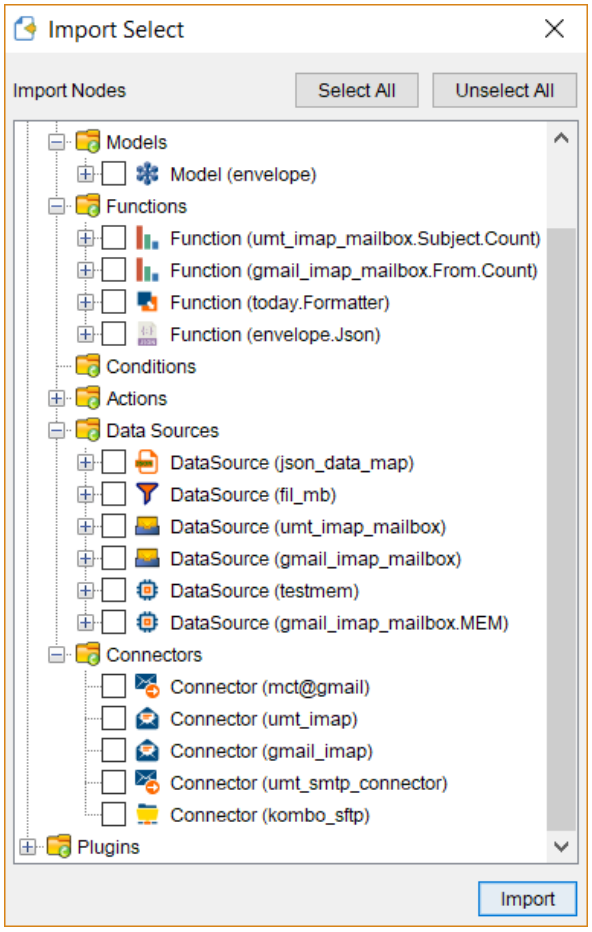


Figure 13 - Import Node Selection dialog

The import node selection dialog shows the node tree of the target configuration in familiar format. User can browse the tree and select the nodes that are to be imported. On selection, PARE Importer auto selects all downstream dependent nodes and on un-select, it auto-unselects all upstream dependent nodes. The Select All and Unselect All buttons exist for convenience.

3.1.8 Toolbar – JMX Remote

PARE engine includes a JMX interface which makes it possible to download and upload entire configuration XMLs remotely. When an engine instance is running with the JMX listeners enabled, using the Remote Button, user can download its currently loaded configuration and load it locally. It is also possible to then upload the same (after, may be, making some changes), which will trigger a remote re-load in the target PARE engine.

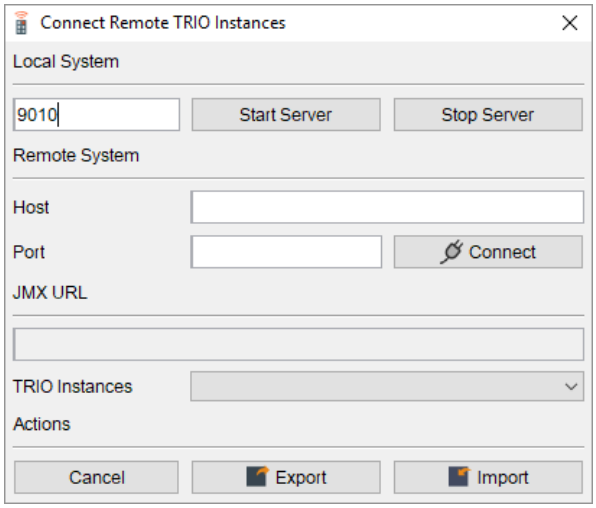


Figure 14 - JMX Controller

Note: As of now, the Import and Export performs a 'replace' of existing configurations and all previous resources are unloaded. So users should be careful before using this feature in production scenarios.

3.1.9 Toolbar – Bundles

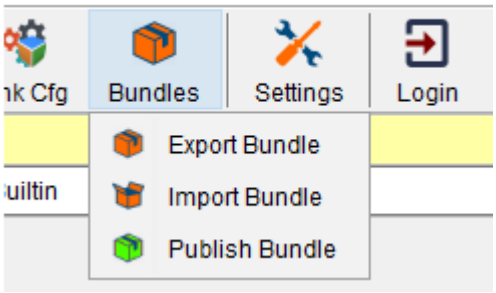


Figure 15 - Toolbar Bundles

3.1.9.1 Export Bundle

The Export Bundle functionality packages the configuration and its dependencies as a ZIP and saves it. On clicking this button, user is prompted for a Bundle Name. The same is used to set the name of the ZIP file.

Exported ZIP file location: {user's home}/PAREExportBundles/{bundle name}/{bundle name}.zip

The Export Bundle feature packages the currently loaded configuration with all its dependencies, including:

- JAR or Class Directories that backs the defined Class Connectors
- Model JAR files

The ZIP file has the following structure:

- {bundle root}/{bundle name}.xml – the exported configuration
- {bundle root}/lib/ - All JARs and/or Class Directories

Note: The export bundle feature should be used to reliably transfer configurations from one environment to another.

3.1.9.2 Import Bundle

The Import Bundle can read the ZIP created via the 'Export Bundle' feature and deploy the configuration in a target machine. On clicking the Import Bundle, user needs to provide path of the Exported ZIP file.

Once the ZIP file is successfully processed, the configuration is available in following location: {user home}/PAREImportBundles/{bundle name}/{bundle name}.xml

User is given option to load the imported XML at the end of successful import.

Note: The export->import bundle method should be used to reliably transfer configurations from one environment to another.

3.1.9.3 Publish Bundle

This is just a combination of an Export Bundle followed by an Import Bundle – combined.

3.1.10 Toolbar – Server Admin

The Server Administrator provides facility to manage a remote PARE Server.

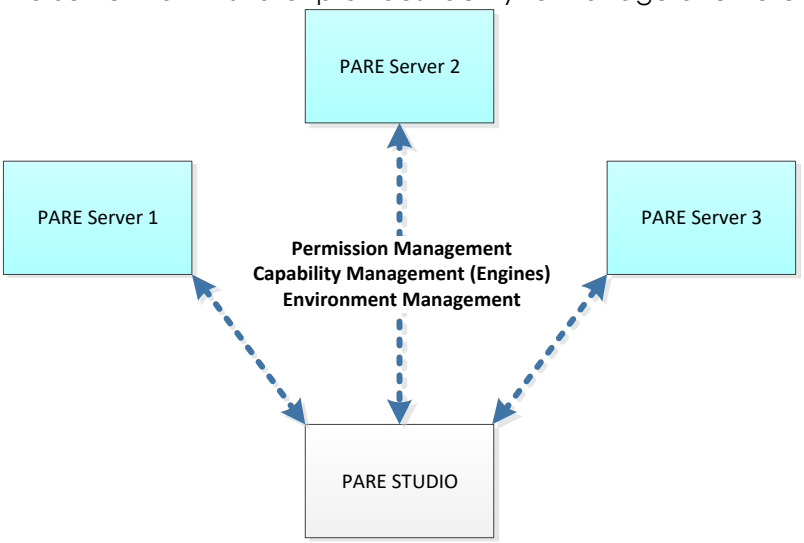


Figure 16 - PARE Server administration

Users can start PARE servers (which are basically PARE configuration containers running on a servlet container like Tomcat) on multiple hosts in their landscape and then centrally manage them using PARE GUI. It's possible to manage the following aspects per Server:

- Users, Workgroup, Assets and access management
- Engine (= configuration bundle) deployment: manage which servers have what capabilities
- Environment management (syncing up the Host parameters, Settings and Named passwords) and Partition management
- Copy configuration from one server to another

To learn more about PARE server, refer to this section. The administrator allows the following functionality:

3.1.10.1 Manage Users

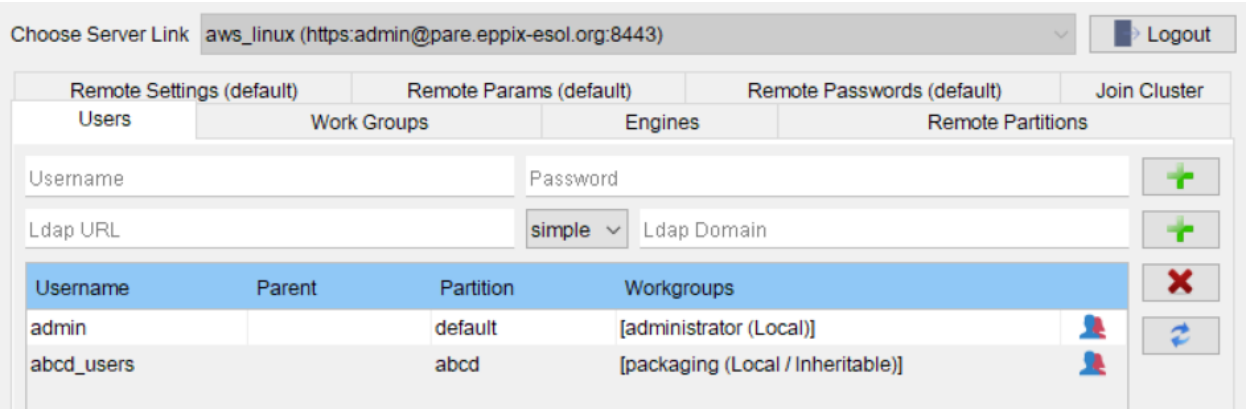


Figure 17 - User manager

- Add or remove users (built-in users like 'admin' cannot be removed)

Once a User is created, you can visit the User properties dialog:

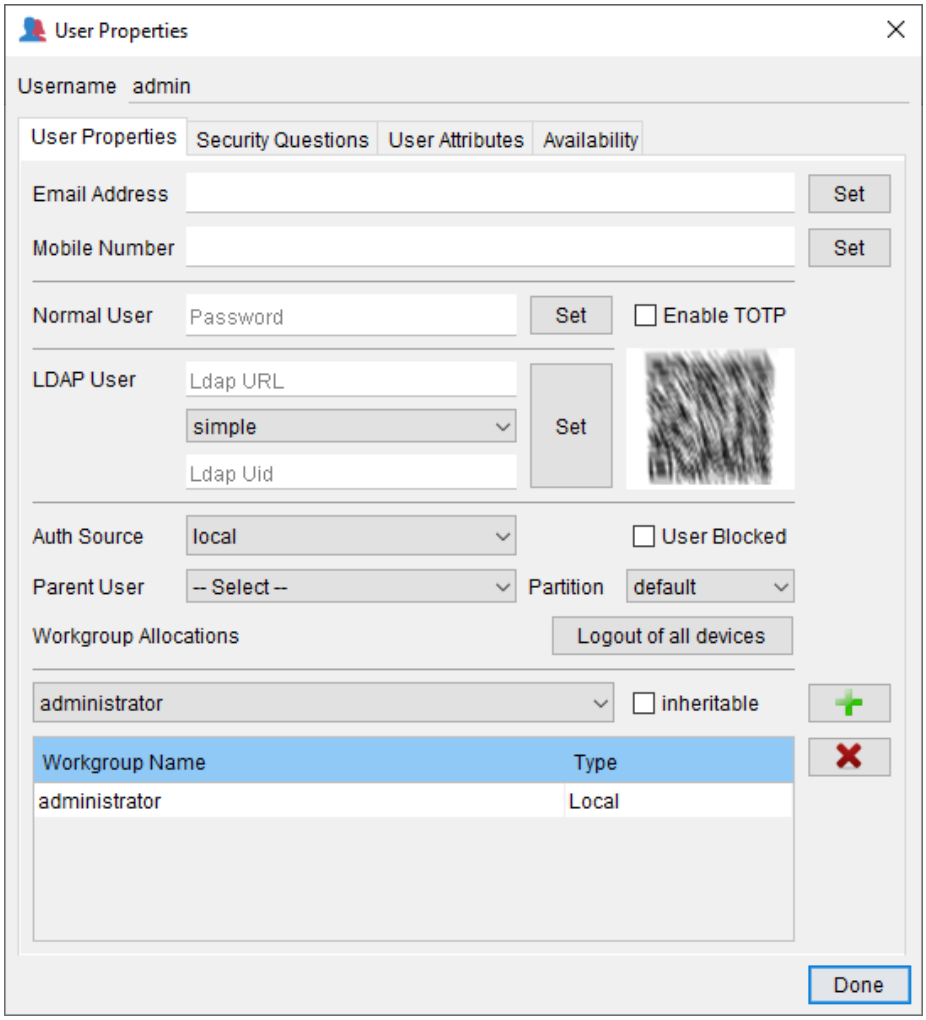


Figure 18 – User Properties Dialog

Here you can:

- Setup email address and mobile number. Note that these can be used as alternate IDs during login and hence need to be unique.
- Set User's authorization source
  - Option 1 – Local

        - EITHER Set a password
        - OR Set LDAP server details
        - PLUS Choose if TOTP based 2FA will be used (note that any standard TOTP client like Google Authenticator is supported)
    - Option 2 – Remote (OpenID Connect Provider)
        - Select a preconfigured Auth Source from the dropdown
- Set user's workgroups:
    - Add a Workgroup
        - Set as Inheritable (optional – will be automatically inherited by child users)
    - Remove a workgroup
    - View all allocated workgroups (locally assigned and inherited)
- Set user's parent; note: the child inherits the parent's 'inheritable' permissions and the parents' partition
- Set user's partition; note: if user's parent is set, the user inherits the parent's partition – which cannot be changed at the child user level
- Logout user from all devices: invalidates all Access Tokens issued against this User, thus forcing a re-login
- Block / Un-block a User

The Security Questions tab helps set the security questions and answers which may be used for password recovery:

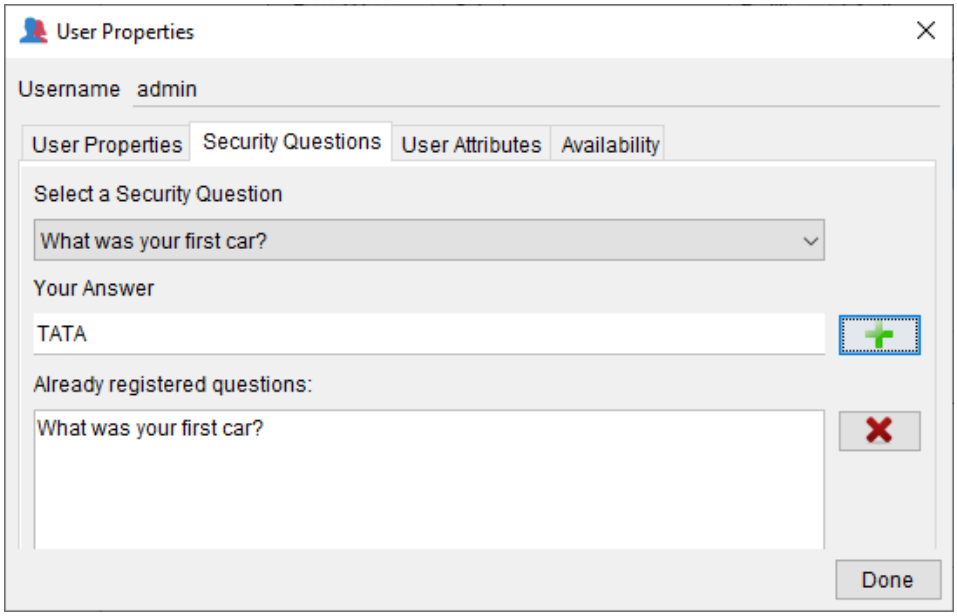


Figure 19 - User Security Questions dialog

Here you can:

- Setup answers to as many security questions (and answers) for the User. The Questions are fixed – can be adjusted as per project requirement.
- The configured questions and answers can be used to recover a lost password. Note that this only works when Authorization Source is Local and LDAP option is not being used.

Note: This feature is in testing and even though API support is available, Web UI support is in the works.

The User Attributes tab helps set up User Attributes that may help in filtration and as part of the Skill based Task mapping feature of the Workflow Engine.

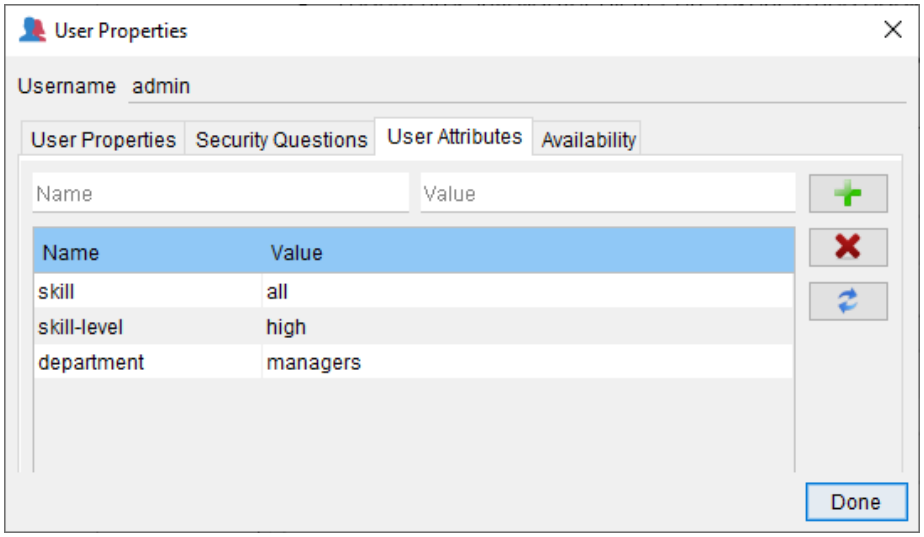


Figure 20 - User Attributes dialog

Here you can setup user attributes (key-value pairs) which can used for following purposes:

- To indicate skills and skill levels of users
- To indicate other functional associations like department, roles, any contractor associations, or any other criteria

The Availability tab lets you define the Holiday Calendar and registered leaves of a user:

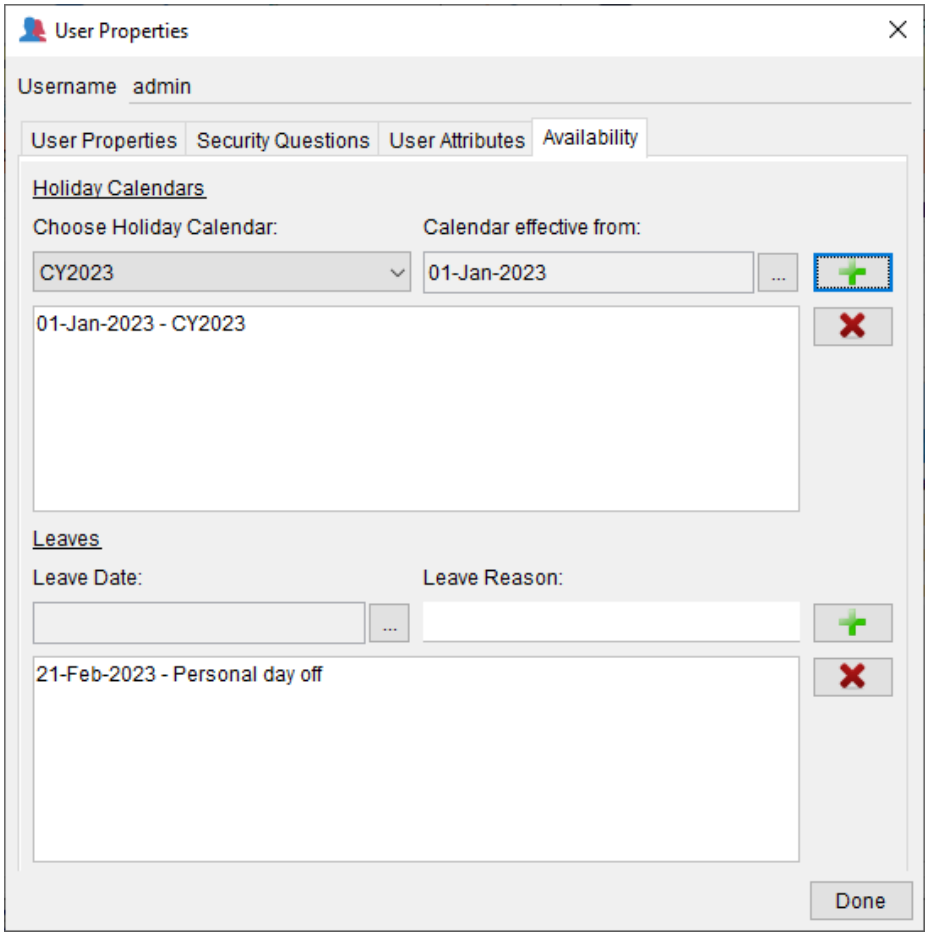


Figure 21 - Availability configuration

Here you can setup the User's holiday calendar (note that multiple Holiday Calendars can be configured in the Settings Area of the Studio). User's leaves can be recorded here as well. This

information is used by the PARE Workflow Engine to check availability and auto-assign tasks to only available users.

Note: User management operations can be done either by the super user (for any user defined across partitions), partition administrators (for any user within the partition) or any other user in the parent hierarchy of this user.

3.1.10.2 Manage Workgroups

Workgroup is synonymous with a Role. It specifies privileges including what type of API from what engines can be accessed by a User assigned with a Workgroup.

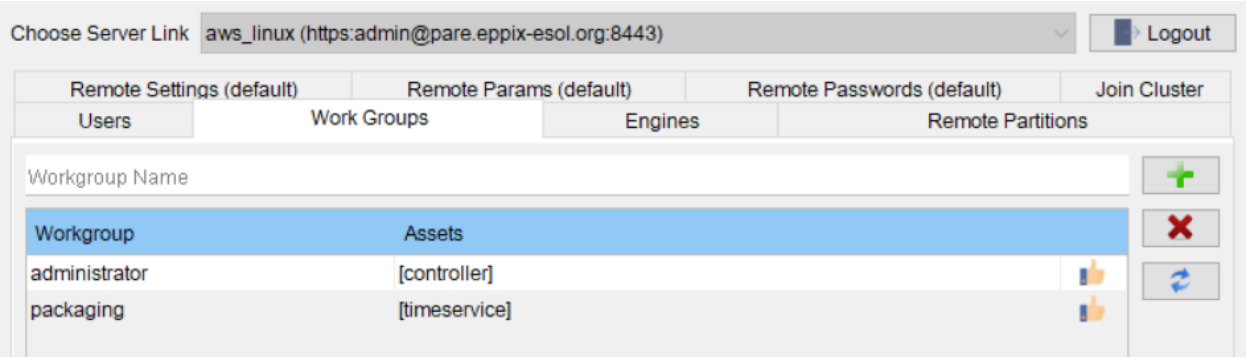


Figure 22 - Manage Workgroups

Adding new Work and removing existing Workgroups can be done from above screen. Once a Workgroup is created, you can set its properties using the dialog below:

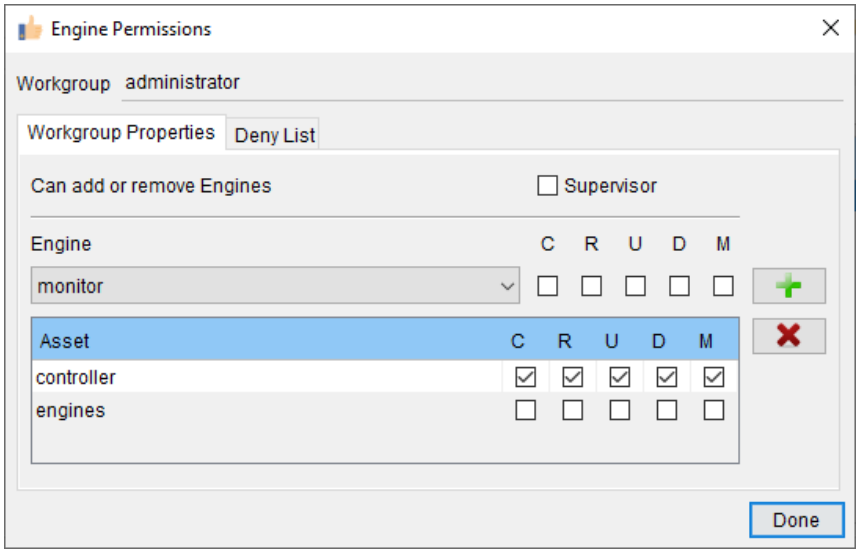


Figure 23 - Set engine permissions

Meaning of CRUDM permissions:

C - Create access – can call POST APIs of an engine
R - Read access – can call GET APIs of an engine
U - Update access – can call PUT APIs of an engine
D - Delete access – can call DELETE APIs of an engine
M - Manage access – can launch Processes of an engine

Multiple engine permissions can be assigned to a Workgroup, thus effectively controlling what parts of the Application any user with the Workgroup can access.

The Deny List tab allows us to configure if a particular API of an engine is denied. This helps achieve fine-grained access control.

Note: Workgroups are system wide definitions and therefore only the superuser can create, manage, or remove workgroups.

3.1.10.3 Manage Engines

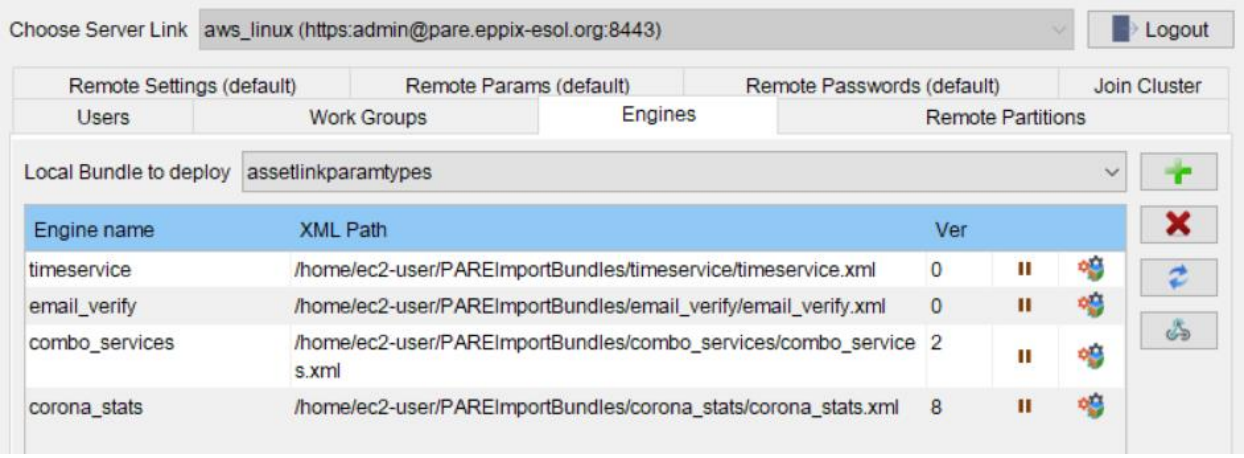


Figure 24 - Engine manager

An Engine is a functional module within the PARE Server. Essentially, it's a PARE Configuration that is deployed in the Server and its exposed assets are available as HTTP API.

The engine manager enables remote deployment of PARE configurations. It uses PARE Bundling features to implement the same. To know more about PARE Bundling feature, refer to this section. All locally exported Bundles are available for deployment. A PARE Configuration Bundle is called an 'Engine' in context of PARE Server. Functionality:

- Deploy Engine: Deploys an Engine and makes its API available for calling
- Un-deploy Engine: Removes an Engine from the Server
- Pause / Resume an Engine: Pausing an Engine temporarily disables all API exposed in that Engine, without doing an un-deployment

3.1.10.4 Registering Webhooks

PARE allows embedding of Event Actions within process flows, and each Event Action can host one or more dynamically registered webhooks. At run time, whenever an Event Action is executed, it calls the corresponding webhooks (which are HTTP callback URLs), thus enabling event driven integration between disparate applications.

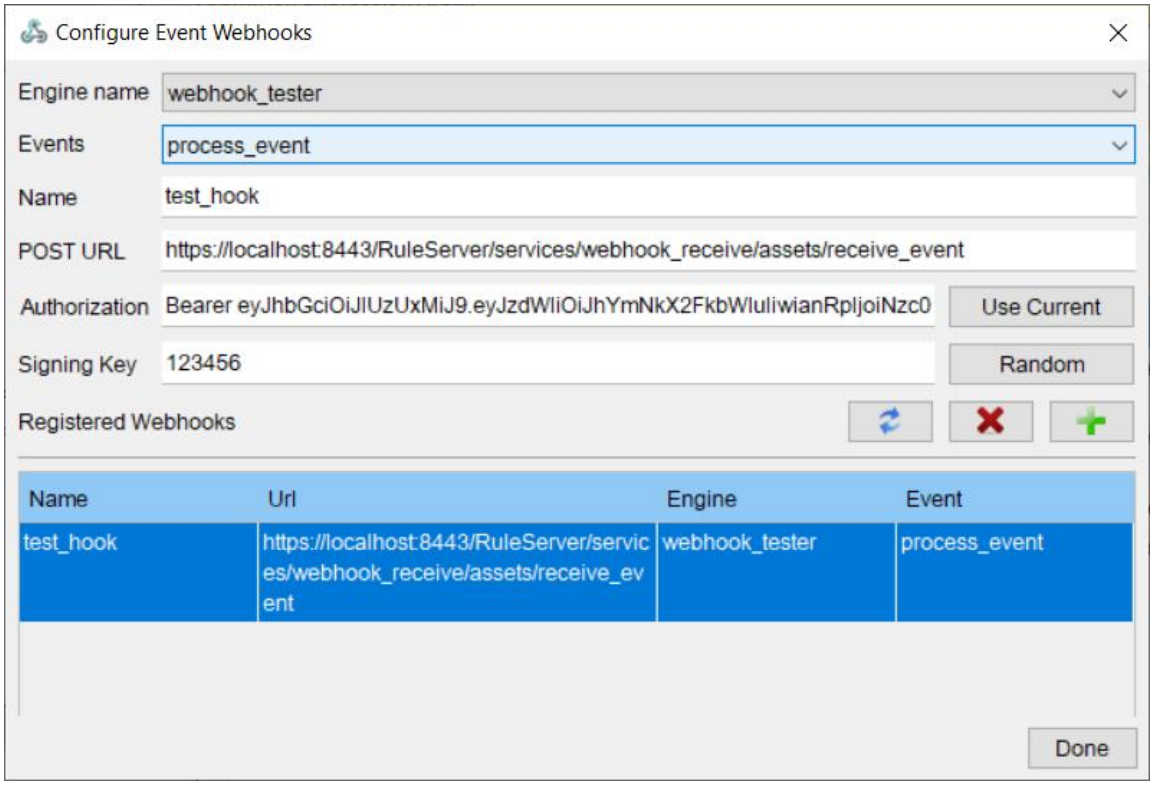


Figure 25 - Creating a Webhook

Upon selecting an Engine, all the events will be listed. Users can now enter the details of the callback URL along with security attributes.

Engine Name	Name of the Engine for which webhooks are to be registered.
Events	Event Actions that are defined in the selected Engine. If there are no Event Actions in the Engine, an empty dropdown will be shown.
Name	Name of the webhook – serves as a search key and this name is used to later remove or modify the webhook.
POST URL	The callback URL. **Note**: Event Actions use the POST method to call the webhook URL and use a single JSON payload to pass event related information. The structure of such information is detailed in the description of Event Actions section later in the document.
Authorization	Here the user can set the content of the Authorization request header. This can be Custom header (like an APIKEY) or a Bearer Token (in case a PARE API is being used for Server Callback.
Signing Key	When a webhook is called by the Event Action, it signs the payload with a signature block in the PARE-Signature HTTP request header. This Signature block is a one-way encrypted hash of the payload itself and can be used by the receiver to verify authenticity of the payload (the receiver must have the same key, of course). The Signing Key is used to calculate the encrypted hash of the webhook payload.
Registered Webhooks	Tabular View of all the Webhooks registered against a particular Engine-Event.

3.1.10.5 Asset Operations

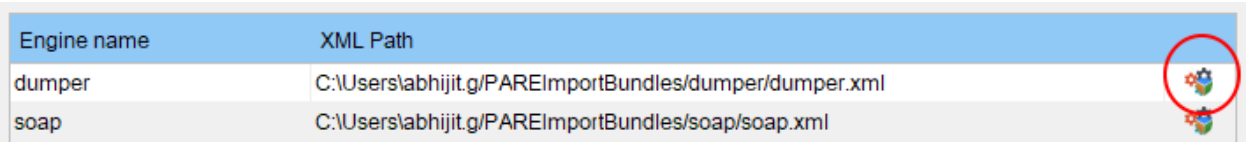


Figure 26 - Open Create Asset Link dialog

On clicking the icon, the Asset Management dialog opens as below:

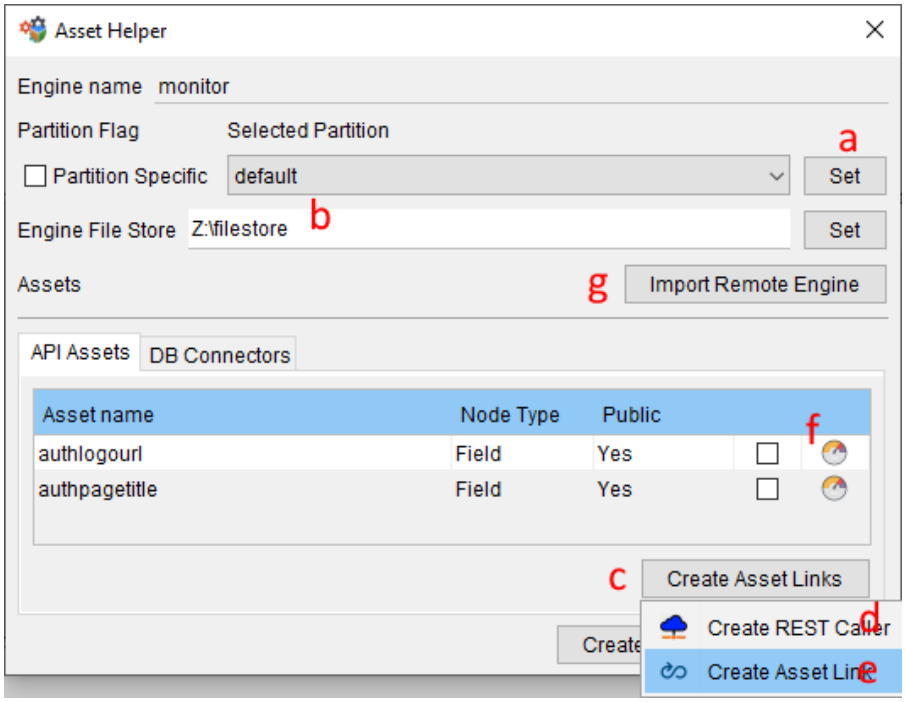
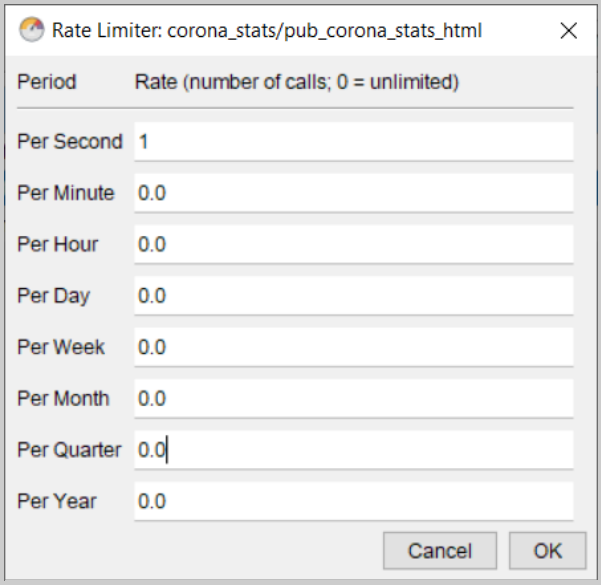


Figure 27 – Asset properties dialog

This dialog lists all the API endpoints that the engine exposes and provides following functionality:

a	Set the engine partition when marked as a partition specific engine. A partition specific engine is only active in the configured partition. Only the superuser can set this flag and data.
b	Sets the engine file store – a directory where the engine specific DMS file content will be stored. This is optional and is always partition specific.
c	Shows menu options: • Create REST Caller • Create Asset Link
d	This option creates REST Service Functions (RSF) for each selected API end-point.
e	This option creates an 'Asset Link' instead of an RSF. While an RSF uses a HTTP connection to call an API, an Asset Link initiated an inter-module call within the same server context if available. More on this later in this document..

f	PARE Server's built-in Rate Limiter:  **Figure 28 - Rate Limiter Configuration** The Rate Limiter ensures that the rate of calling an API is throttled as per the specs above. PARE uses the 'token-bucket' algorithm to throttle API calls. If the call rate goes above any of the limit buckets, a 'too many requests' HTTP error will be issued. **Note**: In a Cluster environment, this may not have desired effect as calls are distributed randomly to a PARE Server pool. Hence, in such scenarios, Rate Limiting should be implemented at the Load Balancer (Reverse Proxy) level.
g	Imports the remote engine and loads it in the Studio. This feature can be used to transfer and load remote configurations locally for making changes and re-deploy.

3.1.10.6 Manage Remote Settings, Host Parameters and Named Passwords

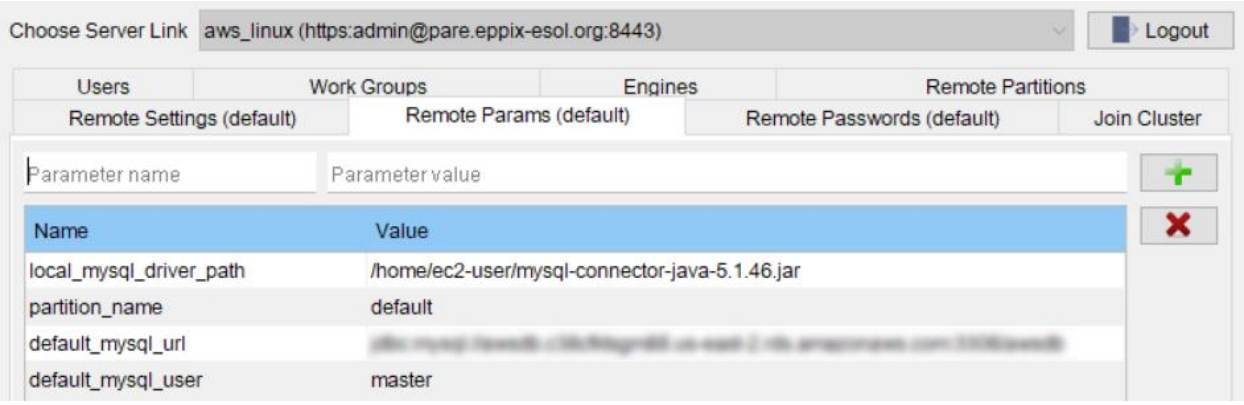


Figure 29 - Manage remote parameters

The remote parameter and password management feature enables setting of Host parameters and named passwords in the environment where the server is running. New parameters can be added, or existing parameters modified or deleted. The following approach is however recommended:

- Do a 'Sync OUT' to copy local parameters (or passwords) to the server
- Edit the parameters (or passwords) individually to set the correct values

<u>Note about Cluster setup</u>

In a cluster setup, if the local PARE Studio is already connected to the Cluster, remote Parameter settings for whole Cluster can be done locally from the Settings area. However, the Remote Settings method is recommended as it automatically acts on the correct partition.

Note about Partitions
The Server Admin screen always shows partition specific information. The partition information is taken from the logged in User properties.

3.1.10.7 Manage Remote Partitions

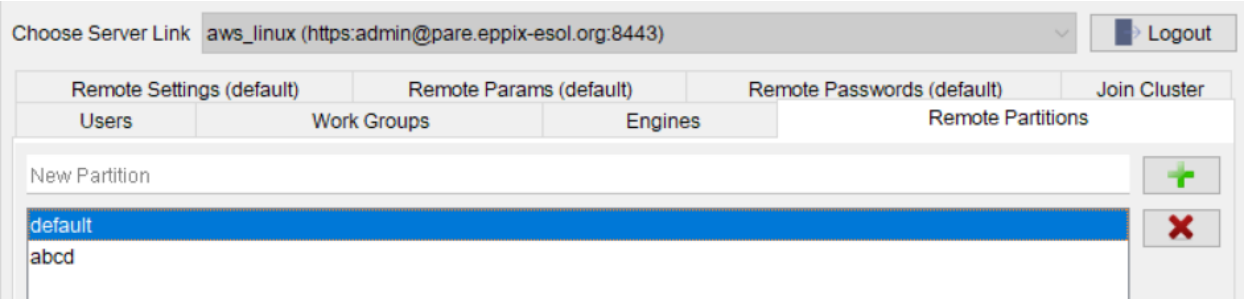


Figure 30 - Remote Partition Manager

The Server Admin screens can be used to view, add, and delete remote PARE partitions. Note that in PARE server all Partitions remain active simultaneously. However, the Server Admin screens show data from the remote partition that is mapped to the currently logged in user.

3.1.10.8 Join Cluster

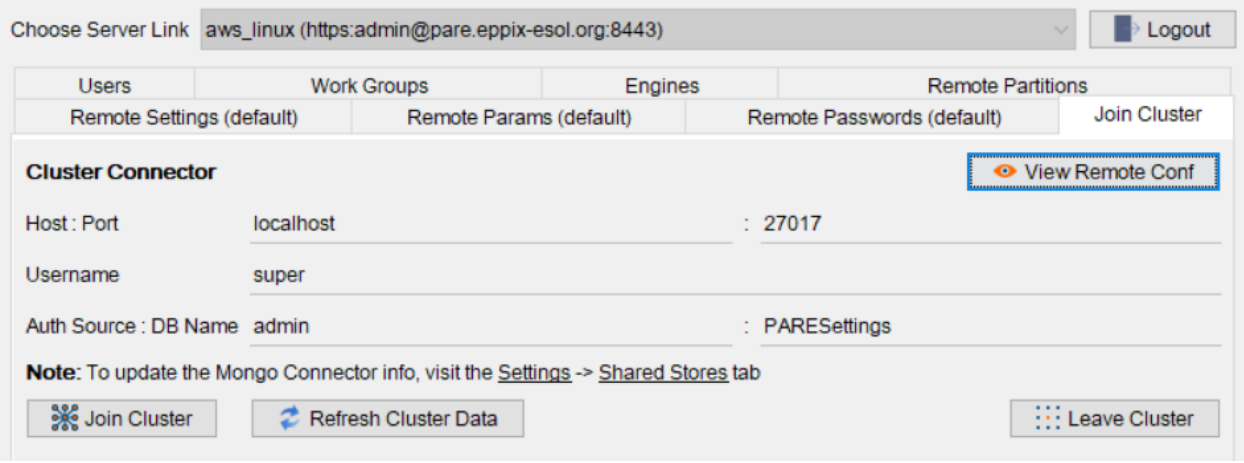


Figure 31 – Joins the remote Server to a Cluster

About PARE Cluster
PARE Cluster is a pool of PARE Servers sharing the same Server configuration (partitions, settings, parameters, engines and so on). The Server Pool is driven by any 3rd Party HTTP Reverse Proxy (like F5, HA_Proxy, nginx, AWS load balancer and similar) which uses the pool to provide load balancing, URL based routing and automatic fail-over.

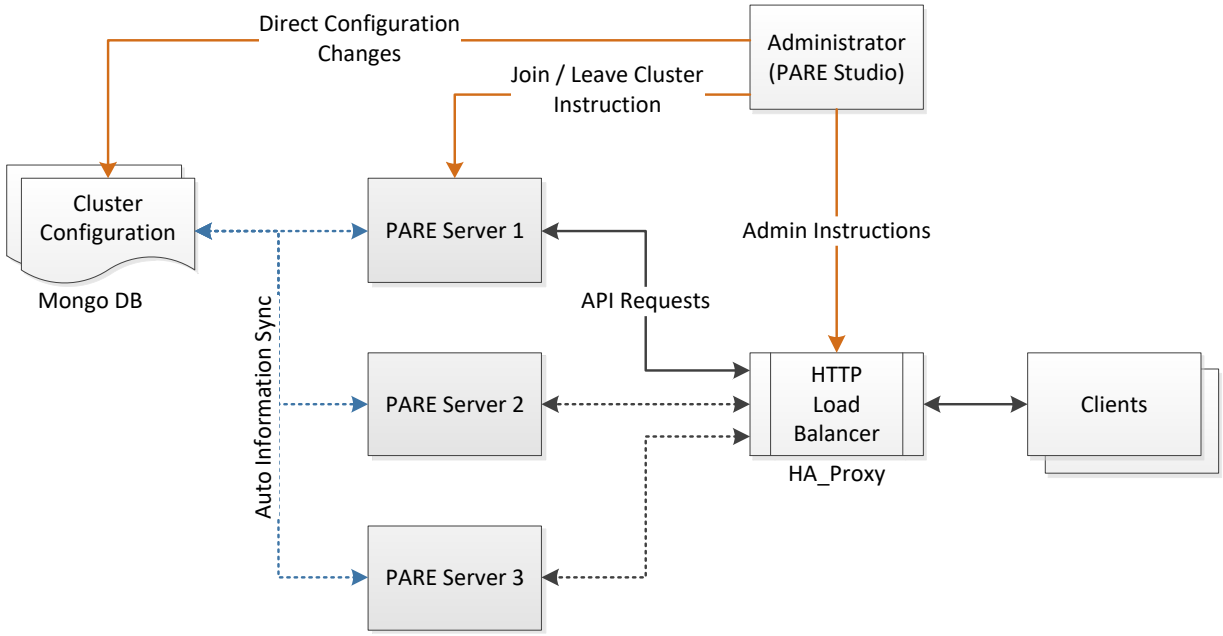


Figure 32 - PARE cluster overview

PARE has 2 modes of operation:

- Local mode – where all environment level configurations are stored in local files (this is the default mode)
- Cluster mode – where all environment level configurations are stored in a shared Mongo DB database, and the PARE Instance is configured to connect to this Mongo DB instance to get and set all environment parameters

A PARE Cluster consists of a pool (one or more) of PARE Servers in 'Cluster' mode and connected to the same MongoDB database, thus sharing the same environment configurations. PARE Cluster allows you to quickly add capacity to serve increasing API call loads and offers Load Balancing and High Availability through a HTTP Load Balancer.

This facility manages the mode of the connected PARE Server:

- Join Cluster – Switches the Server to Cluster mode and configures it to connect to the same MongoDB database that the Studio is currently connected to. Obviously, the Studio must be connected to a valid Cluster database for this to work.
- Leave Cluster – Switches the Server back to 'Local mode'.
- View Remote Configuration – Shows the current Cluster Configuration of the remote server.

3.1.10.9 Remote Files (DMS)

This Tab shows DMS entries per engine and allows operations.

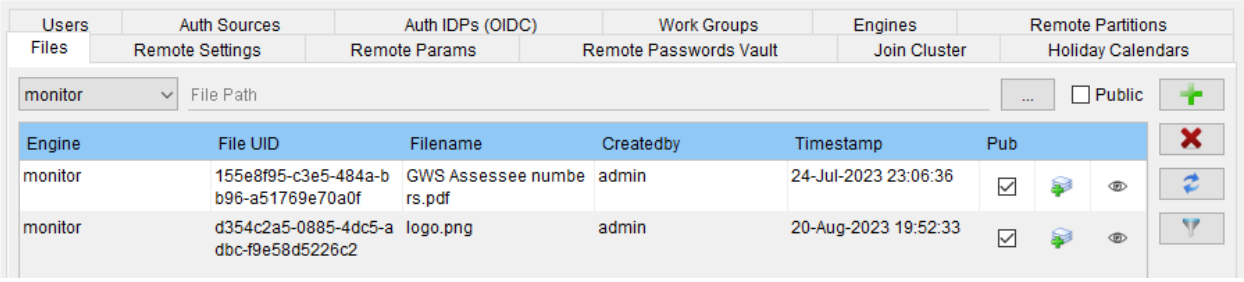


Figure 33 - Remote Files manager

The operations allowed include:

- View file listing per engine

- Insert new file
- Delete an existing file
- Set file attributes
- Filter view based on file attributes

3.1.10.10 Auth Sources

Auth Source configuration allows recognition of external ID tokens (typically acquired through OpenID Connect protocol). This feature allows connecting PARE server authorization to external IDPs like Azure AD, Google and any other IDP supporting OpenID Connect. As of now this configuration is system-wide, i.e. all partitions share this configuration.

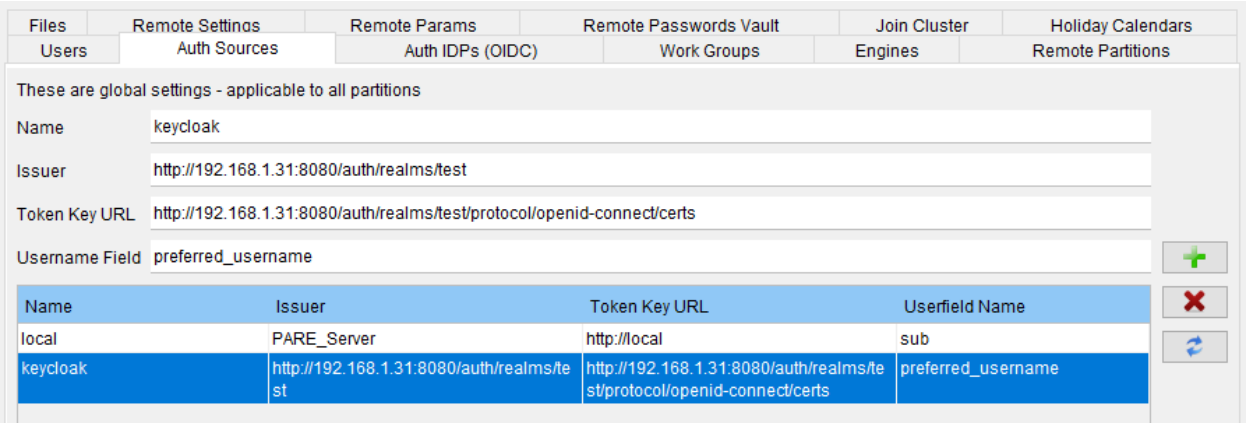


Figure 34 - Remote Auth sources

The configuration includes:

Name	Name of the Auth Source for internal reference. The same name is used to set the Auth Source in the User properties dialog.
Issuer	Issuer of the token – this field helps uniquely identify the Identity Provider. Typically this is the "iss" field of the JWT ID token.
Token Key URL	URL of public keys that can be used to validate the JWT signature. Typically, multiple such public keys are used, and each token may be signed using one of the keys available from this URL.
Token Validity	Number of hours a PARE access token remains valid by default.
Username Field	Name of the field within the token that represents the Username. This should match with an existing PARE User and that user should have the correct Auth source configured so that the Issuer and Key validations can be successfully done per token.

3.1.10.11 Auth IDPs

This Tab is used to configure the Oauth2 authentication flow for each of the defined Auth Sources. The IDP definitions are partition specific (unlike the Auth Source definitions). PARE Server's built-in Authentication web flows use the IDP configurations to redirect the browser as per the OAuth2 "authorization_code" flow to securely acquire an ID Token.

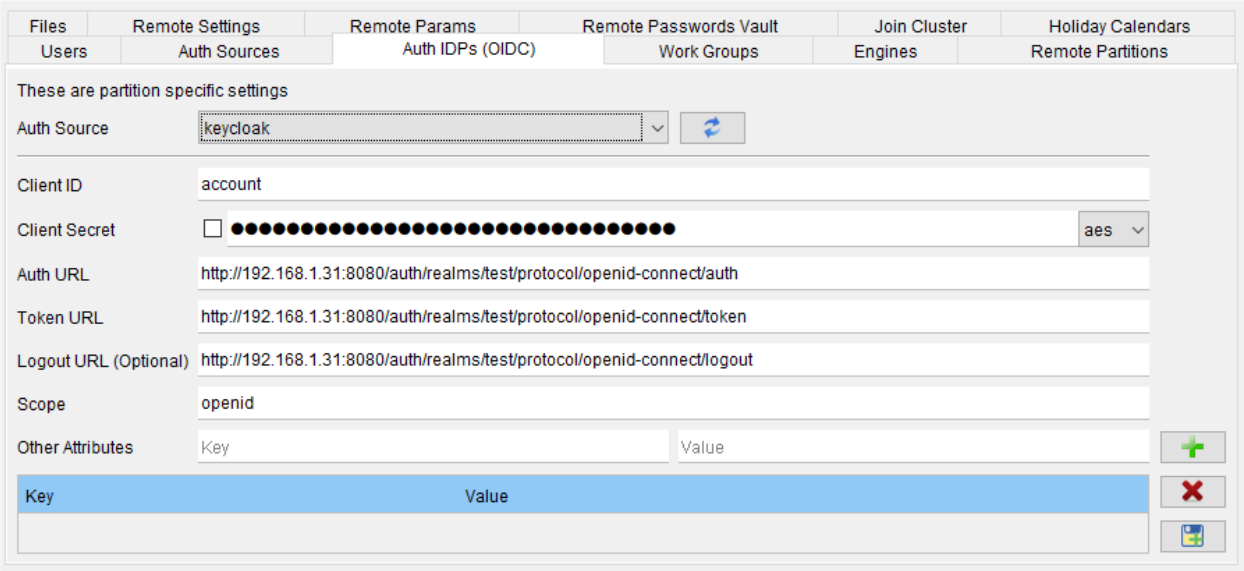


Figure 35 - Auth IDP configuration

The IDP configuration includes:

Auth Source	Name of the Auth Source for internal reference. This is selected via a dropdown.
Client ID	Client ID to be used. This should be available from your account with the Identity Provider. The process may differ for each provider and provider specific documentation may be referred to. To know more about the flow, please refer to this link: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow
Client Secret	Client Secret to be used. This should be available from your account with the Identity Provider. The process may differ for each provider and provider specific documentation may be referred to. To know more about the flow, please refer to this link: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow
Authorization URL	The Authorization URL where the Authentication web flow will redirect on demand. Note that this happens only if the access and refresh tokens are expired.
Token URL	The URL through which the ID token is retrieved, using the authorization code, upon successful authentication.
Logout URL	URL to execute a IDP level logout, which essentially invalidates the ID token and refresh token thus forcing a re-authentication next time.
Scope	Typically includes flags that indicate that an ID Token is requested instead of an access token. These flags are specific to each profider.
Other attributes	This may include special instructions (key-value pairs) to the provider – for example, include a refresh token on each request, or do not include a refresh token at all etc. These instructions are specific to each provider.

3.1.10.12 Remote Settings

This Tab allows manage Remote Server settings.

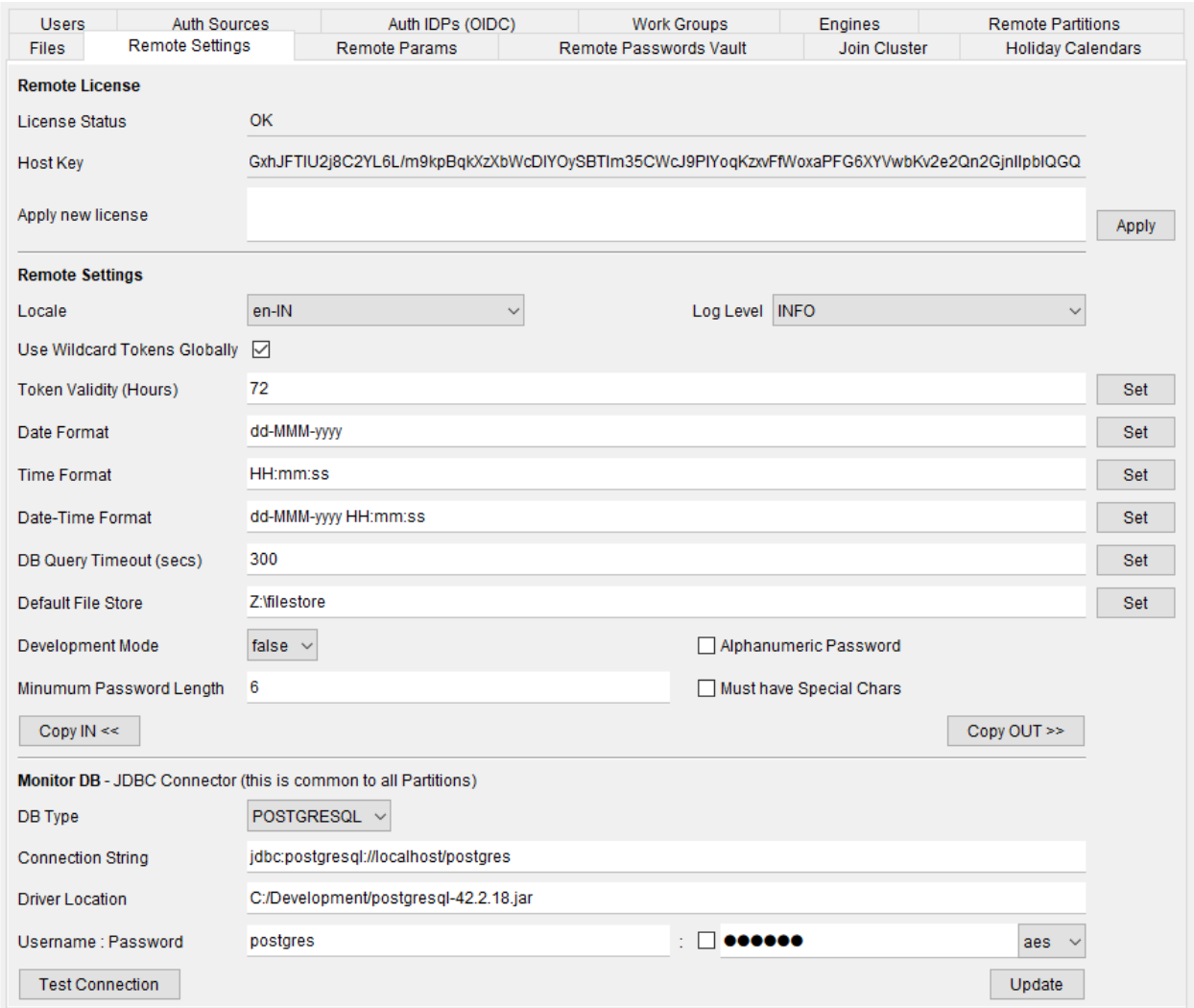


Figure 36 - Remote Settings panel

This screen allows remote management of a variety of server features including:

Remote License	Allows updating a new license remotely. Please note that this feature will work if the server has a valid license. i.e. planned license upgrades are allowed. Once the server license becomes invalid, user will need to apply license locally.
Locale and other standard settings	Most of the settings seen in the local Settings tab can also be set from here – for the remote server and applicable for the logged in user's partition. This includes Locale, Date Formats, DB Query timeout.
Wildcard tokens	By default, PARE access tokens implement a source validity check, i.e. the user's device IP address is embedded in the token and validated in each subsequent request. However, for some use cases, that default behavior may be changed using this flag.
Token Validity	Number of hours a PARE access token remains valid by default.
Default File Store	The default location where all DMS files will be stored. This can be overridden by the Engine level file store configuration described earlier in this document.
Development mode	Flags the server to alter its default behavior including allowing non-secure token cookies.
Password policy	Basic password policy settings.
Monitor DB	Configure the remote Monitor DB connection parameters.

3.1.10.13 Holiday Calendars

This Tab allows configuration of one or more Holiday Calendars.

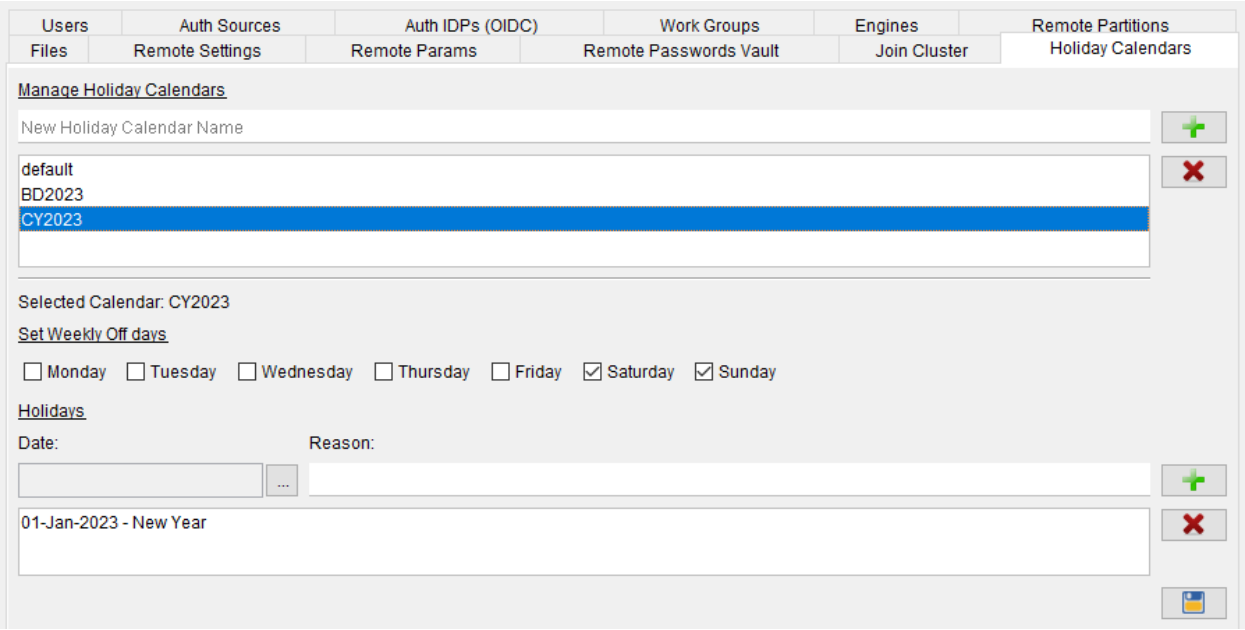


Figure 37 - Configure Holiday Calendars

Following operations are possible:

- Adding a new Holiday Calendar
- Remove an existing Holiday Calendar
- Set Calendar properties:
  - Weekdays and weekends
  - List of Holidays with a description for each holiday

Holiday Calendars defined here can be assigned to an User in the user's Availability configuration.

3.1.11 Toolbar – Settings

For more details, please refer to this section.

3.2 Using the Tree

The left navigation tree is highly responsive, and almost all operations of the Studio start from there. The tree responds to both Right and Left clicks.

On Left click the related tree node is selected and the corresponding Editor form is displayed in the right pane. For, example, in Figure 3, when Built-in Function (subString) is selected the corresponding Form is automatically displayed in the right pane.

Right clicks can be done at 3 levels:

- Top level (at Rules Config root node)

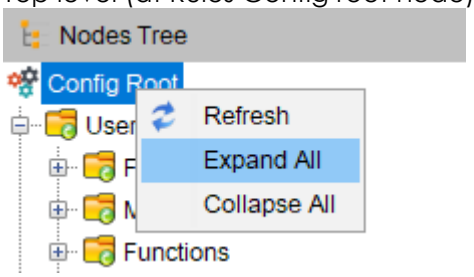


Figure 38 – Root menu

- At the Node Type Level

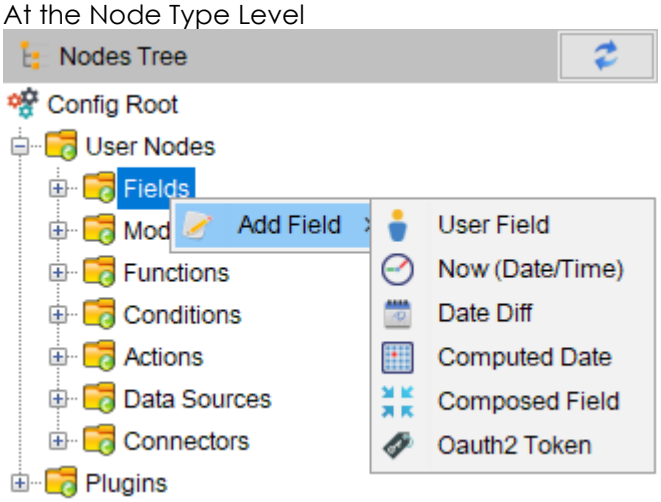


Figure 39 – Node Category menu

- At the Node Level

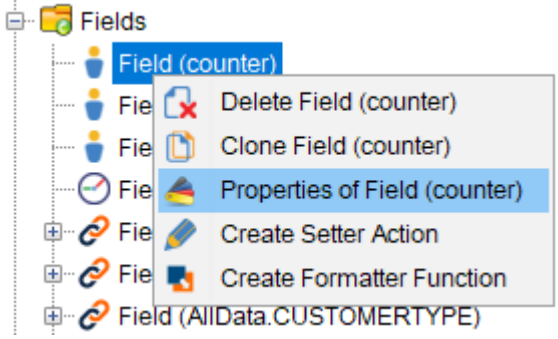


Figure 40 – Node menu

3.3 Expressions

Expressions are one of the most flexible and powerful features in PARE. It basically enables embedding of other PARE resources as a STRING. Expressions are used throughout PARE configuration and are the reason for much of its flexibility.

Expressions consist of following components:

Fields: Any Field can be embedded within PARE expression by putting its name inside square brackets. For example, a Field with name 'abc' is represented in PARE expression as '[abc]'.

Functions: Any Function can be embedded within PARE expression by putting its name and parameters inside curly brackets. For example, the 'substring' function can use embedded within expression as '{substring(Abhijit, 4, 0)}'. Note that, other Functions and Fields can be placed in place of the Parameters.

Static Text: Static Text is not replaced. Hence an expression 'Short name is {substring(Abhijit, 4, 0)}' will get evaluated as 'Short name is jit'.

Parameters: Parameters within container Actions or Functions can be accessed within expressions using '[parameter name]'. Where-ever expression editors are there and parameters are applicable, hitting Ctrl-SPACE will bring up the expression component drop-down, which includes the list of parameter symbols that can be inserted in the cursor position.

Escape Sequences and double quotes: '\' is the escape sequence and is used to exclude parsing of the next character in the expression. "" are used to exclude parsing a range of

characters in the expression. For example, if the first parameter of Builtin.substring function contains a comma, then we need to either escape it or put it within double quotes so as not to confuse the parser:

```
{substring(a,bc,   2, 0)} - wrong - will result in parameter mismatch
{substring(a\,bc,  2, 0)} - right - returns 'bc'
{substring("a,bc", 2, 0)} - right - returns 'bc'
```

There are various flavors of expressions as below:

Simple expressions:

These are expressions, where PARE resources are substituted. When user is just evaluating expressions by calling PAREReplacer.getInstance().replace(), this is what happens.

Before Evaluation: Hi [customer name], \n Your payment due date is [bill due date]. \nThanks, Admin.\nDate: {timestamp()}

After Evaluation: Hi James Long, \n Your payment due date is 31-Dec-2020. \nThanks, Admin.\nDate: 02-Aug-2016

Note: In case data-source fields are used, that the value of the Fields and Functions are calculated based on current data-source cursor position.

Computable expressions:

If a numeric context is available for the expression, it is automatically computed using the built-in math parser in PARE. A numeric context is available when the expression is associated with a Return type or an Input parameter type. So, for example, a Composed Field of INTEGER return type, will automatically try to use the math parser to evaluate the expression. Similarly, any parameter of the Numeric type uses the math parser to compute the value of the expression. Following math operators are supported:
+, -, /, *, ^, %
Example: {maxusage([customer number], [date of usage])} / [call count]

Condition expressions:

If a BOOLEAN context is available to the expression, it is automatically evaluated using the built-in comparison parser in PARE. A BOOLEAN context is available when the expression is associated with a Return type or an Input parameter type of BOOLEAN. This is also done in case of Expression type Conditions (and if condition expressions are embedded within Rule and Decision Table actions). The expression is used to arrive at a TRUE/FALSE conclusion. Following conditional operators are supported:
>, >=, <, <=, ==, | |, &&
Example:

- ✓ ([customer age] >= [senior citizen age]) && ([call count] > 0)
- ✓ '[data.billingname]' == '[param_name]'

Note:

- ✓ Condition Parser supports numeric data, and hence 2 == 2.00 will return TRUE and 2 == 2.01 will return FALSE. You can also embed computations inside a Condition expression.
- ✓ If Strings are to be compared, a single quote (') needs to be provided on either side of the comparison operator.

Expression Helper:

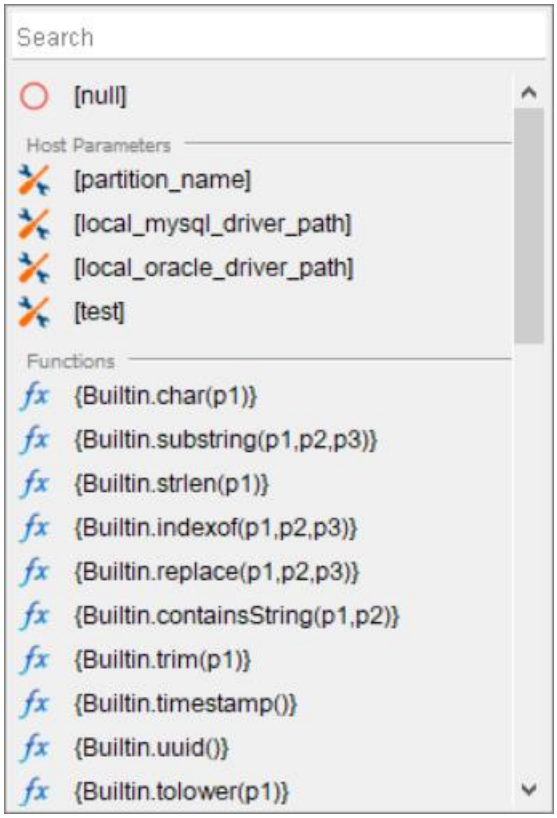


Figure 41 - Expression Helper popup

Expression Helper is a popup menu that shows re-usable expression components (Fields, Functions, Parameters …), that can be inserted in the Expression Editor area of the PARE GUI. This can be invoked in any cursor position within the editor by hitting Ctrl-SPACE. The selected item of the popup menu will be inserted within the editor text. The facility is available in all places where PARE expressions can be entered. The relevant editor fields can be easily identified by their yellowish background.

The Popup menu supports search – typing a few letters in the search box filters the list, making it easy to find intended nodes.

3.4 Script Support

PARE uses J2SE Script Engine feature (JSR 223) to support embedded scripting. Currently three languages are supported:

- JavaScript
- LUA
- Python

This feature enables definition of scripted algorithms for designing user generated Functions and Actions. PARE automatically exposes existing PARE assets to the Script environment, including:

- Fields (all or selected)
- Functions (all)
- Actions (all)
- Parameters (all)

The execution is flow is like:

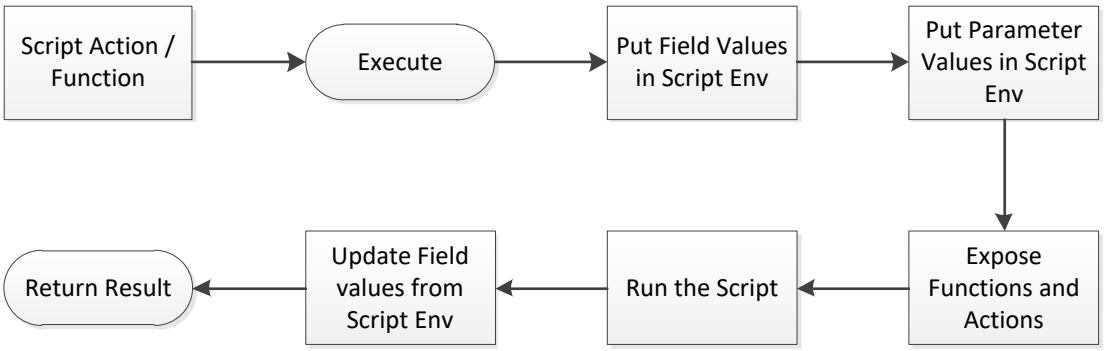


Figure 42 - Script execution flow

The various PARE assets available to the Script are easily accessible to the user by means of a popup menu like below:

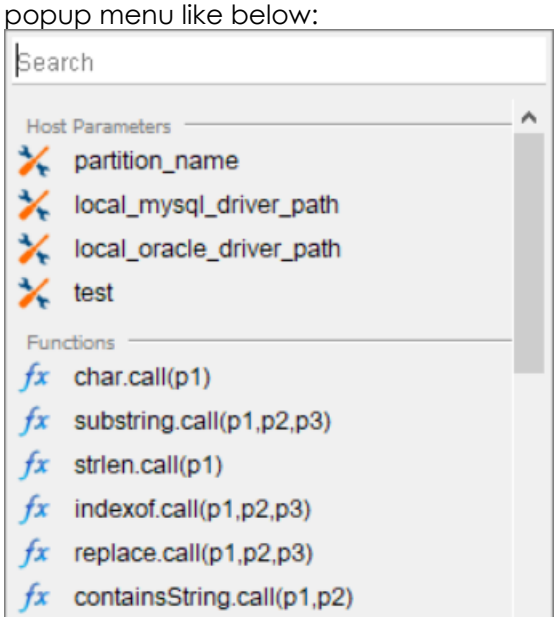


Figure 43 - Script popup menu

Parameters: Parameters are accessed by names. Parameter values can be changed within the Script, but that has no impact outside (scope of the Parameter values are inside the Script only).

Fields: Existing PARE fields can be directly accessed within a Script. However, these are copies of the actual fields, and hence changes to the field values are not automatically propagated outside of the Script boundary. Please refer to the Utils paragraph above to understand methods to sync the field values inside and outside of the Script.

Functions: The underlying object for all PARE functions are exposed automatically. For ease of use, a special 'call' method is available (which is included in the Helper dropdown). Note that due to language specific features, the .call method is applicable to JavaScript only. For LUA, a special :luacall('…') method is made available. Since LUA does not support Java style array Var-args, LUA scripts should embed the parameters within a string. For example, calling 'substring' in JavaScript with a field 'X' looks like: substring.call(X, 2, 0); whereas in LUA the equivalent is: substring:luacall(X..', 2, 0');.

Actions: The methodology is same as Functions as described above. The only difference is, Actions do not return any value.

PARE GUI includes a syntax aware editor that highlights language features and PAREnthesis so that users find it easier to code within the GUI.

3.5 PARE Settings and Password Vault

PARE settings and password manager functionality helps in environment localization.

For example, the PARE configuration XML needs to access a DataBase table. But, the database access parameters are different in Production and Test environments:

- Database server is different
- Username is different
- Password is different

In such cases, it's not convenient to store these environment specific data in the Configuration XML, as different versions of the same configuration will need to be maintained per environment. Often this is cumbersome and risky as the Test Configuration may be accidentally pushed into Production environment or vice versa causing data corruption.

Therefore, the DB URL, Username and Password are stored in Settings – which is a one-time setup per environment. Thereafter actual configuration XML refers to the Settings parameters instead of hard-coded values. Hence, the configuration XML can be shared un-changed across different environments safely.

3.5.1 Instance and Host Settings

These settings define some common characteristics of the Instance (goes into the configuration XML) and Host (gets stored at machine level outside of the configuration XML).

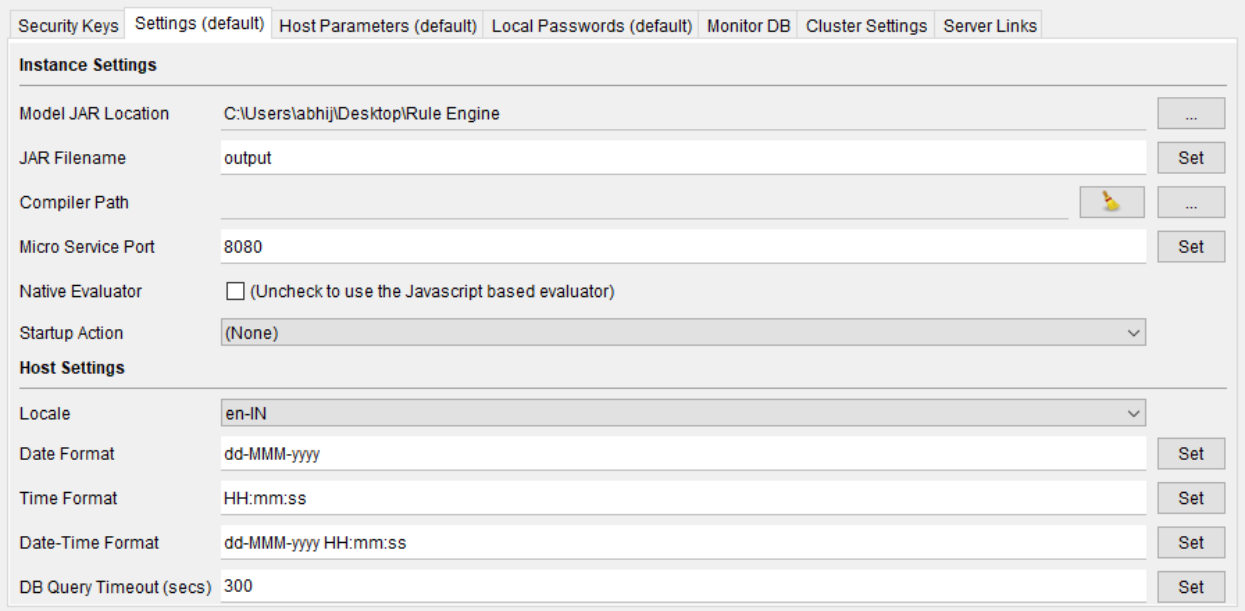


Figure 44 - Instance and Host Settings

Model JAR Location	Local directory where the Model JAR will be created. This setting is stored in the configuration XML. The default directory is the same as location of RuleEngine.jar.
JAR Filename	Name of the JAR filename. The final path = <JAR Directory>/<JAR filename>.jar. The default name is 'output'.
Compiler Path	Creation and deployment of Models require that a compatible Java compiler is available to PARE. In case the engine is invoked from an installed JDK, this is not required. Else (when invoked via a JRE) and model deployment is required, a valid path to javac.exe needs to be provided.
Micro Service Port	The local listen port used by the PARE Micro Service (deprecated)
Native Evaluator	This flag is used to determine if the JavaScript engine will be used to

	evaluate 'condition' expressions. If the flag is 'checked', a built-in engine is used, else the JavaScript engine is used. If JavaScript engine is used, some syntax compliance is required in the expressions.
Startup Action	Here one Action can be selected that is automatically fired when the configuration is first loaded. This Action must have have ZERO parameters.
Locale (per partition setting)	Select the appropriate Locale settings. All JVM supported Locales are available. **Note**: This setting may alter behavior of number formats – like interpretation of .(dot) and ,(comma) characters in numbers.
Date/Time formats (per partition setting)	Default input and output formats of Date/Time fields. For available formats refer here.
DB Query Timeout	Default timeout for all DB Queries – unless overridden in the query configuration.

3.5.2 Host Parameters

This feature offers creating one or more name-value pairs that are stored outside of the configuration and remain accessible to ALL configurations in the environment where the settings are defined. Settings can be accessed just as Fields – via PARE Expression. The Ctrl-Space popup will always show the defined settings.

These Host Parameters can be used in place of the Database Server URL (in the above example), so that the XML configuration can be used unchanged in both Test and Production environment. **Note**: Both environments must have the same parameter names defined with values that are specific to the environment.

Note: Host Parameters are linked to the PARE Partition. Hence different partitions may have different values for same Host Parameters.

Host Parameters can be accessed from the Toolbar, and the UI is like below:

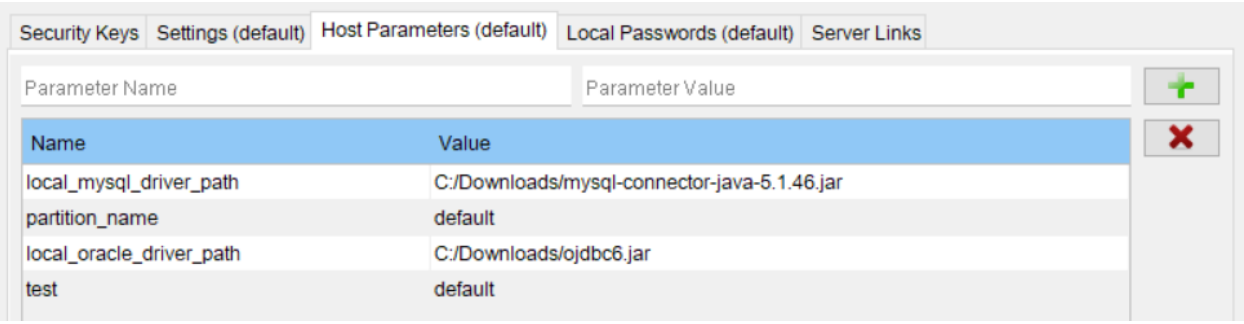


Figure 45 - Settings editor

The Expression popup menu shows the defined Host Parameters like below:

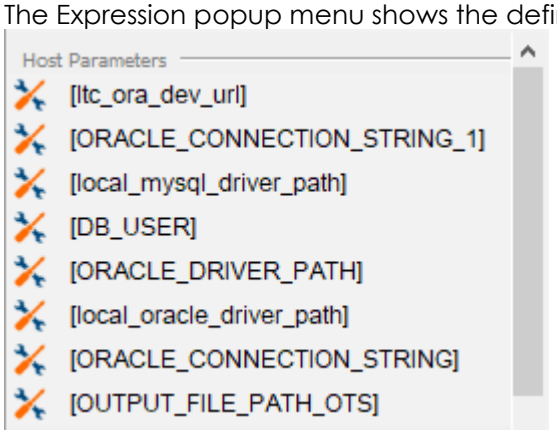


Figure 46 - Expression popup

3.5.3 Password Vault

PARE Password Manager provides a password store where multiple named passwords can be saved in encrypted format. Password Manager is mandatorily required by all the relevant Connectors (DB, SMTP, FTP, POP/IMAP). This provides additional security as the Configuration XML never contains any password. Instead it contains a name – the actual password is supplied by the Password Manager, based on the name, at run-time.

The Passwords can be set using the UI below:

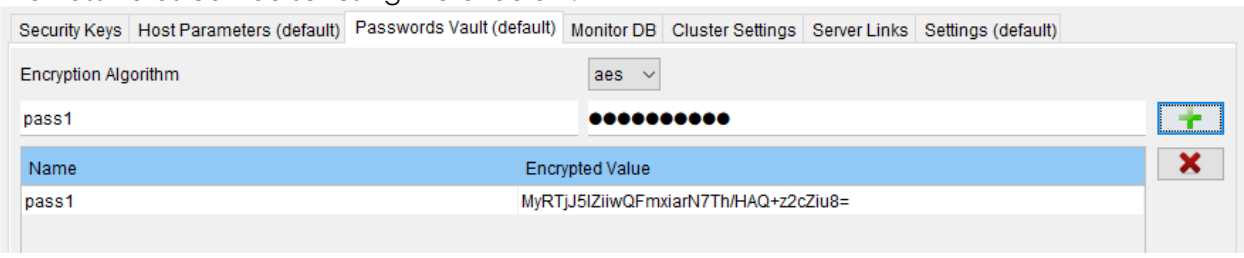


Figure 47 - Password store editor

Note: The default encryption algorithm is AES/256

Subsequently, for example, a DB password field can be linked to a saved password using the Ctrl-Space popup menu like below:

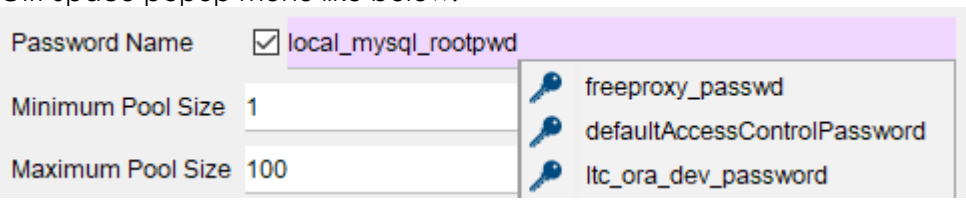


Figure 48 - Using the Password Manager

Note: PARE Passwords are linked to the PARE Partition. Hence different partitions may have different values for same Password name.

3.5.4 Server Links

PARE Studio helps manage one or more PARE Servers (including user management, access control and engine deployment). Server Links define the Server location (host/port) and login credentials that will be used to login for administering a server.

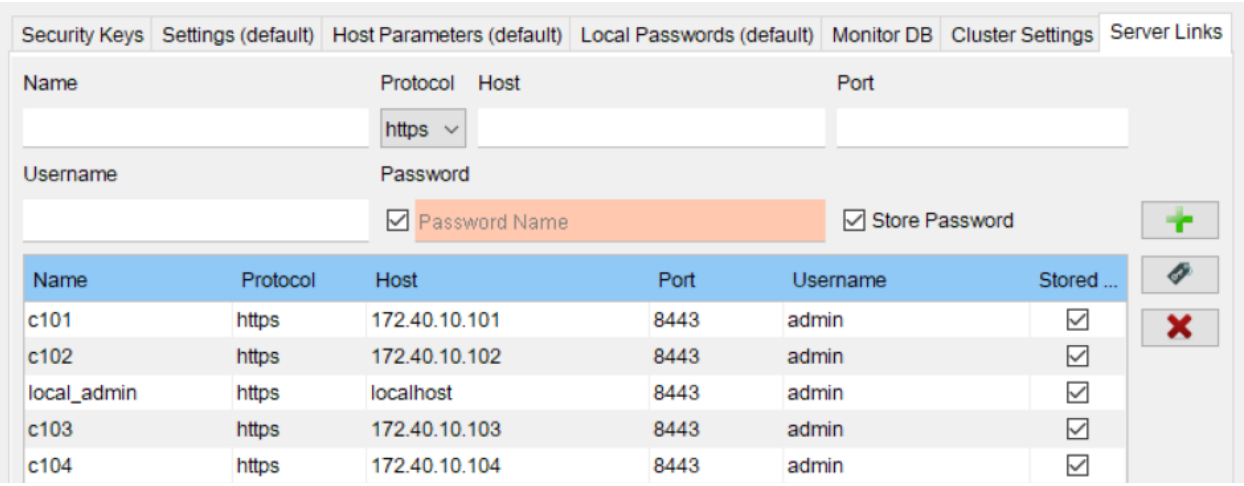


Figure 49 - Server Link editor

Protocol	Whether the server is using SSL.
Name	Unique name of the server link – used to select the correct link in the

	Server Admin screens.
Host	Host name or IP address of the PARE Server you want to manage.
Port	Port to which the Server is listening.
Username	Login Username. Note: This user must be part of the 'administrators' workgroup.
Password	User's password. You can use a named password from the password manager, or type in directly.
Store Password	If this is not Checked, the password is not stored as part of the Server Link and when the Link is used to Access any remote Server, a password input dialog will appear.

These links are stored in the Settings area and are used for:

- choosing the server to administrate
- push new Host Parameters to a remote server
- push new Passwords to a remote server

Note: Server Links along with Server Admin features enables deployment of capabilities across your infrastructure and management from a single GUI.

3.5.5 Security Keys

PARE lets you set custom encryption keys for various encrypted content that is creates or uses. Depending on whether its running on "Local" or "Cluster" mode, the encryption keys are stored either in a local KeyStore file or as an encrypted object in the shared Cluster database.

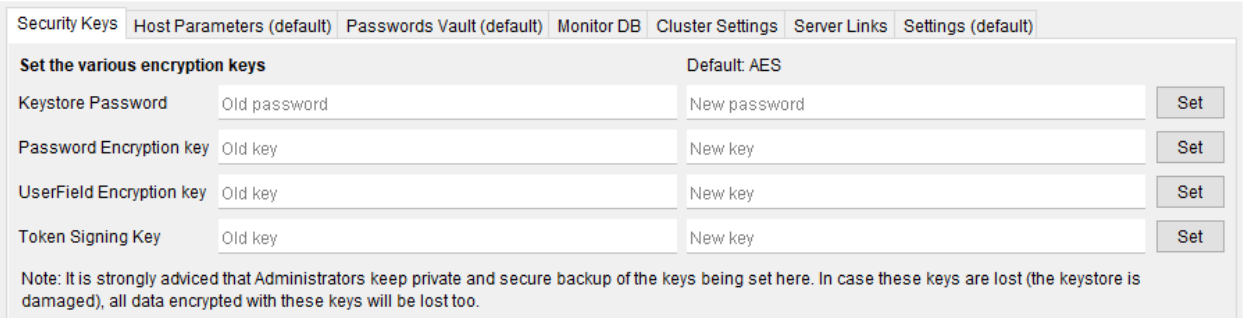


Figure 50 - Security key setter

Default Encryption	Shows the default encryption used to encrypt the vault. The default is always set as AES/256 and its strongly recommended to keep this setting.
Keystore Password	Local mode: Password required to access the KeyStore. Cluster mode: Key used to encrypt the Keys object in Cluster DB.
Password Encryption key	Encryption key used to securely store the local Passwords (refer here). If this key is changed, the local passwords are automatically migrated. Note that if the Password key is changed, existing configurations with directly entered passwords encrypted using the previous key will no longer work (a blank String will be returned on decrypt operation). In that case the password will need to be set afresh.
UserField Encryption key	Encryption key used to store value when a User Field is marked as 'encrypted'. Note that if the User key is changed, existing configurations with Field values encrypted using the previous key will no longer work (a blank String will be returned on decrypt operation). In that case the field value will need to be set afresh.
Token Signing Key	Custom key used to sign the OAuth2 tokens. Note that once this key is changed, all Tokens issued so far are invalidated. **Note:** 1 – Changing the Token Signing key globally invalidates all tokens. Before

	taking such a drastic step, consider if "Logout User" will be enough for your requirements. 2 – The Token signing key should be changed (like set to something qualifying as a strong key) during the initial setup process.

If you don't fully understand what you are doing, it's best to leave these keys untouched. In case you want to reset the keys, please contact support. In case you forgot the old keys, you have set, contact support. Please be aware that if you change and then forget the value of the keys, you will lose the data encrypted with those keys.

3.6 Fields

To create a new Field, right click on the 'Fields' node and select the appropriate Field Type as shown below.

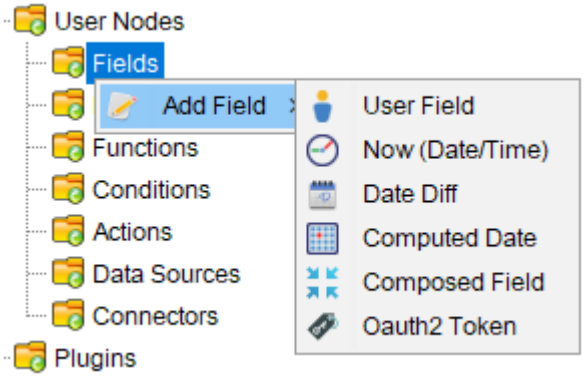


Figure 51 – Add Fields menu

To select an already existing field, just select the corresponding Tree node by Left clicking it. Note that, not all types are fields can be created from the Fields menu. For example, Data Source Fields are created from the right-click menu options of the corresponding Data Source.

3.6.1 Common Characteristics

All Fields share some common characteristics that make it usable uniformly throughout the Engine. Those include:

- Thread local – All fields can be set as Thread Local or Thread Shared. In the former case, a new version of the Field is implicitly created per application thread. For example, same thread local User Field can be used to increment a counter in two parallel threads without any conflict. Updates to thread shared fields are synchronized. Typically, User Fields can be set as Thread-Local true or false. Other field types are always set to Thread-Local true.
- The Studio includes a 'Test' button in all Field editor forms. However, whether any actual data will be shown depends on configuration. For example, 'test'-ing an Object Field with no actual Object map will result in an error.
- All Fields can be embedded inside PARE expressions. Hitting Ctrl-Space when the input cursor is in an expression field, will bring a pop-up menu will all candidates for insertion in the expression. Fields are inserted as **[field-name]** within expressions.
- All Fields have a DATATYPE:
    - STRING
    - INTEGER
    - LONG
    - FLOAT
    - DOUBLE
    - DATE
    - TIME
    - DATETIME
    - BOOLEAN

3.6.2 User Field

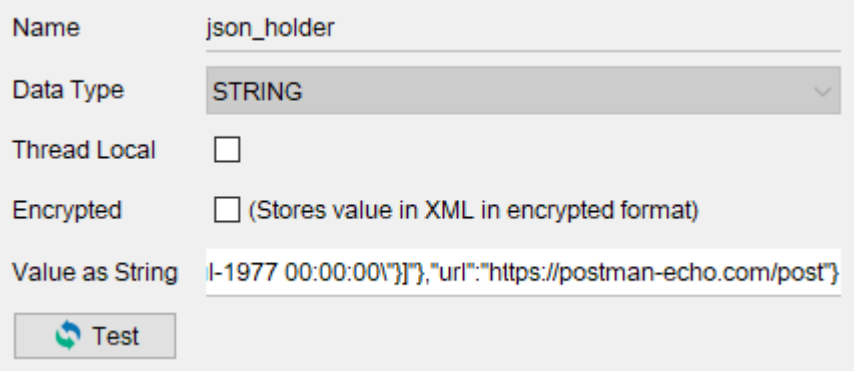


Figure 52 – User Field editor

User fields can be accessed by calling the setValue and getValue API. These are useful to store parameters just before calling an action, for example, or storing static values.

Name	Name of the Field
Data Type	Data type of the Field (STRING, INTEGER, …)
Thread Local	Whether the field value is local to the executing thread. Note: If set to YES, the value of the field will not be shared across threads.
Value as String	Value of the Field in String format – can be entered in the GUI
Test	Shows current value of the Field

3.6.3 Data Source Field

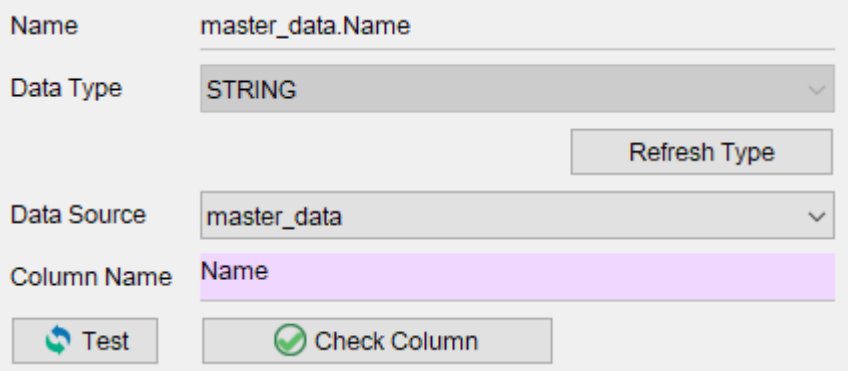


Figure 53 – Data Source Field editor

A Data Source field exposes a data source to the PARE Engine. Currently 3 different data sources are supported:

- RDBMS
- File (Delimited or Fixed Length one record per line)
- Application Objects

Name	Name of the Field
Data Type	Data type of the Field (STRING, INTEGER, …)
Thread Local	Whether the field value is local to the executing thread. Note: If set to YES, the value of the field will not be shared across threads.
Data Source	Underlying data source used to dynamically calculate value of the field
Column Name	Name of the data source column to which the field is mapped. Ctrl-Space brings up a popup menu with list of all available column names.
Test	Show current value of field. Note that the value at current Cursor position of the data source is shown. If the Cursor position is 'before-first', in that case the value at the first row is shown.

Check Column	Since column values can be manually entered, this button enables checking of the column name against the columns defined in the Data Source.

Essentially a Data Source presents data to the PARE Engine as a Table:

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	…
Row 1								
Row 2								
Row 3	Cursor >>			**Field Value**				
Row 4								
…								

Figure 54 – Data Source Field editor

The tabular data can be provided by a DB Table, or a File or an array of Application Objects. The field is always mapped to a particular Data Source and a Column (say Column 4 in above example). The value of the Data Source field depends on current Cursor position. Typically, Data Source Fields are used in the context of a Job which has capability to process a data source.

An example of how a RDBMS Data Source field's value is resolved:

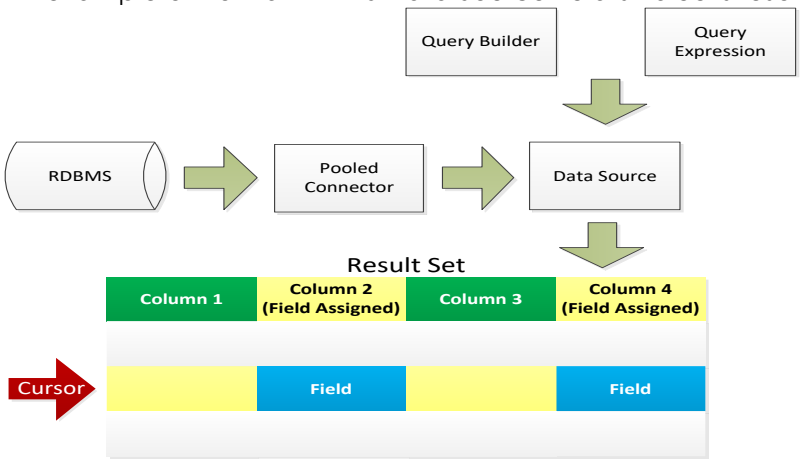


Figure 55 – RDBMS Data Source Field schematic

An example of how a File Data Source field's value is resolved:

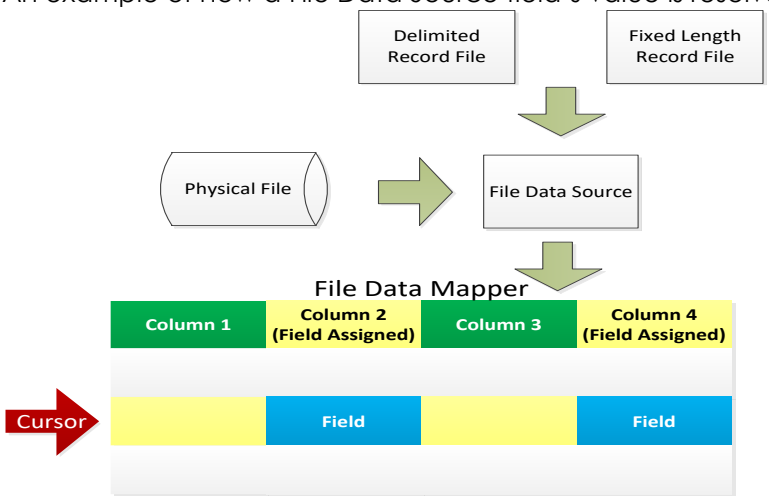


Figure 56 – File Data Source Field schematic

An example of how an Object Data Source field's value is resolved:

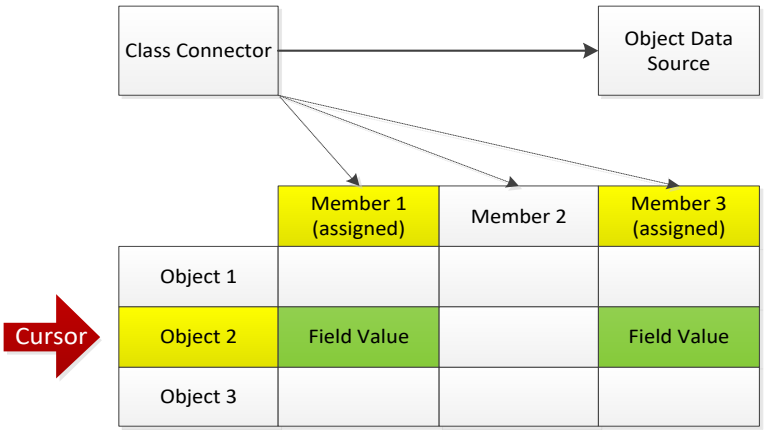


Figure 57 – Object Field schematic

Among the Data Source Fields, currently only the Object Data Source supports setting the value of a field.

Recommended approach to create Data Source Fields is to use the right-click menu on a Data Source node and clicking the 'Create Datasource Fields' option:

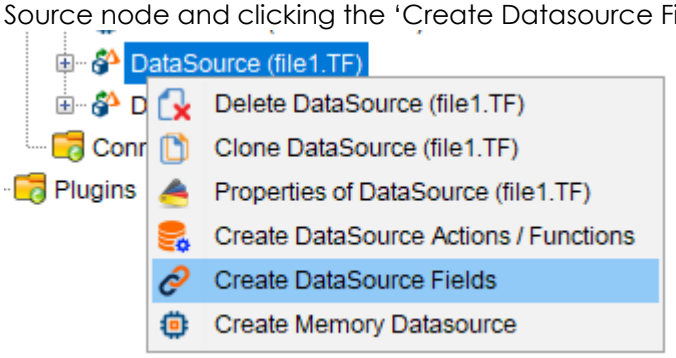


Figure 58 - Create Datasource Fields

This will show a popup window with the list of mapped columns:

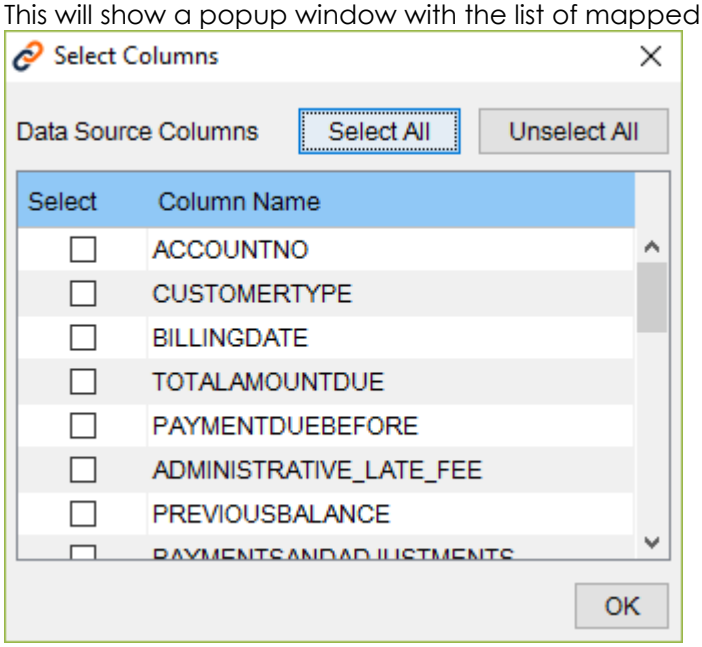


Figure 59 - Select Datasource Columns

Now user can select the required columns of the Data Source, and the fields will be created automatically.

3.6.4 Now (date/time)

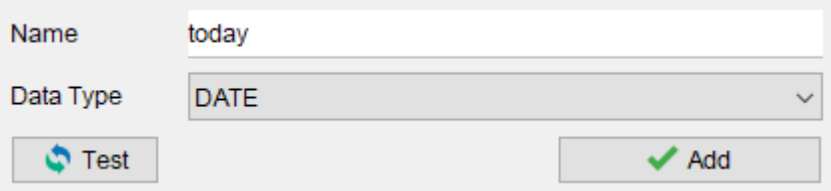


Figure 60 – Now Field editor

This temporal field stores current date time information. Data is accessed as either Date (dd-MMM-yyyy) or Time (HH:mm:ss) or both (dd-MMM-yyyy HH:mm:ss).

3.6.5 Date Diff

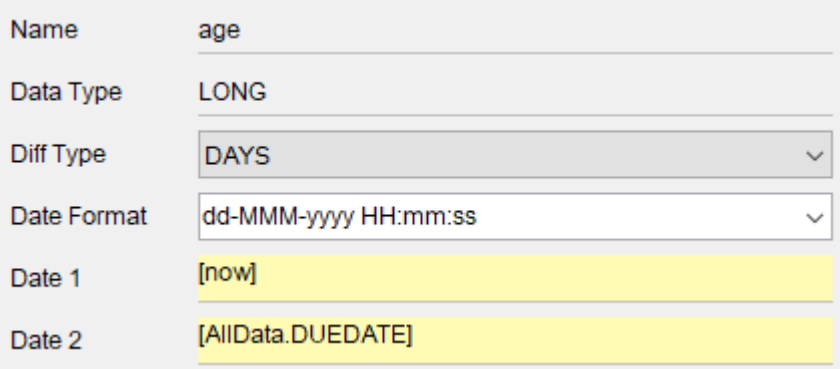


Figure 61 – Date Diff Field editor

Date Diff calculates difference between two Dates in terms of Years, Months, Days, Hours, Minutes and Seconds. The two date fields are expressions so as to allow forming dates as per specified format from various other fields. In the simplest case, just insert two DATE type fields in expression syntax ([within square brackets]).

Name	Name of the Field
Data Type	Fixed to LONG (cannot be changed)
Thread Local	Whether the field value is local to the executing thread. Note: If set to YES, the value of the field will not be shared across threads.
Diff Type	What the value of this field signifies.
Date Format	Input Date format for both Date 1 and Date 2
Date 1	Expression that should resolve to a Date object and follow the Date Format
Date 2	Expression that should resolve to a Date object and follow the Date Format
Dependencies	Shows other fields or functions that are referred in the expressions used in this Field.
Test	Shows the current value of the Field.

Note: All editor panes with expressions have a Dependencies list. It lists all the Nodes embedded inside the expressions. This list is auto-populated when using the GUI.

3.6.6 Computed Date

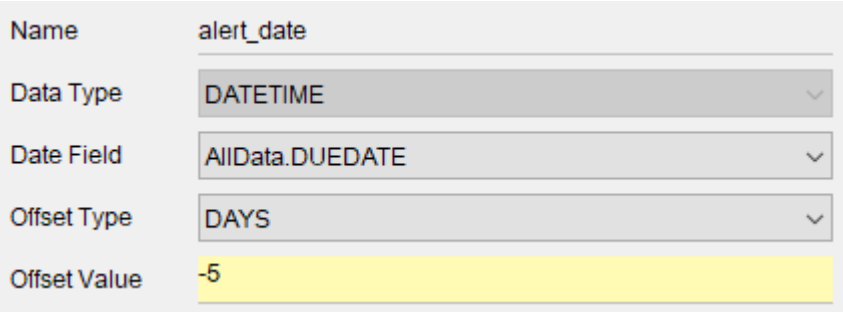


Figure 62 – Computed Date editor

The Computed Date field adds or subtracts YEARS, MONTHS, DAYS, HOURS, MINUTES or SECONDS from a given Date/Time field. The output can be mapped to a Date, Time or Datetime value.

Name	Name of the Field
Data Type	Any one of DATE, TIME and DATETIME
Thread Local	Whether the field value is local to the executing thread. Note: If set to YES, the value of the field will not be shared across threads.
Date Field	An existing field of type DATE, TIME or DATETIME
Offset Type	Offset value in DAYS, HOURS, MINUTES etc
Offset Value	Numeric value (can be positive or negative). The value is calculated by adding the offset (UoM is Offset Type) to Date Field.
Dependencies	Shows other fields or functions that are referred in the expressions used in this Field.
Test	Shows the current value of the Field.

3.6.7 Composed Field

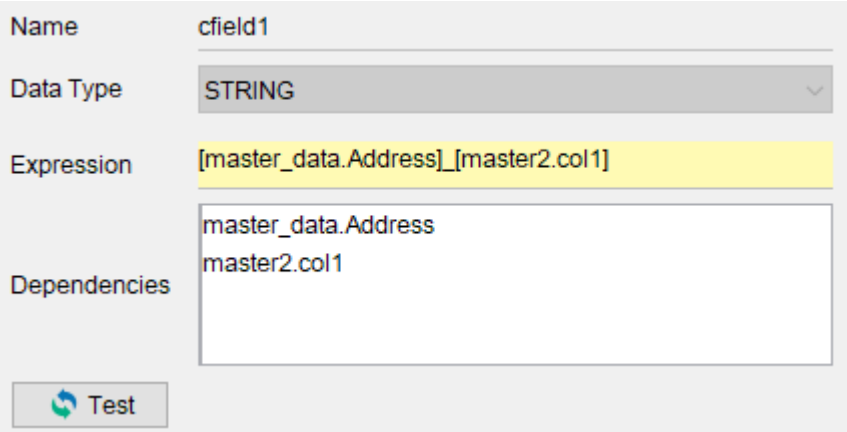


Figure 63 – Composed Field editor

Composed Fields are used to integrate other fields to create a new Field. It's versatile, for example, three integers can be mixed to form a date with a particular format. For numeric fields, it supports arithmetic operators for various operations. Following operators are supported:

+	Add numbers
-	Subtract numbers
/	Divide numbers
*	Multiply numbers
^	Calculate the power, e.g. 2^3 = 8
()	Supports brackets

Name	Name of the Field

Data Type	Data type of the field (STRING, INTEGER, …)
Thread Local	Whether the field value is local to the executing thread. Note: If set to YES, the value of the field will not be shared across threads.
Expression	Expression defining the value of the field. Arithmetic parsing is applied if the Data Type is any one of the numeric types.
Dependencies	Shows other fields or functions that are referred in the expressions used in this Field.
Test	Shows the current value of the Field.

3.6.8 OAuth2Token Field

This field stores Oauth2 Tokens and implements the "authorization_code" (now deprecated) and the "client_credentials" flows to acquire an access token. Oauth2 is a popular authentication and authorization method adopted by most of the modern identity and access management platforms. Oauth2 tokens are granted for specific authorized resources on successful authentication, and the same can be used to call authorized API of those platforms. Please refer to this site for further details. The PARE GUI supports Oauth2 token generation.

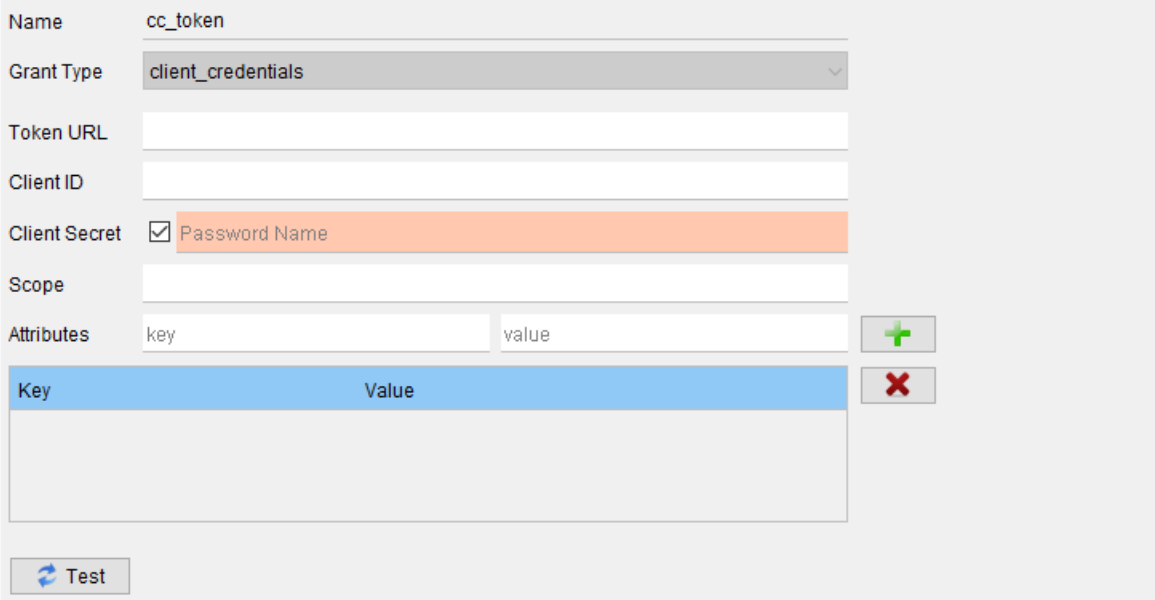


Figure 64 - Oauth2 Token field editor

Name	Name of the Field
Token URL	The Token URL of the Oauth2 provider – typically this gets called with a code returned by the Auth URL, from within the callback service
Client ID	Oauth2 client_id. Various providers have their own custom methods of generating this information. For example, in Facebook, you will need to create an Application, where you get this information.
Client Secret	Oauth2 client_secret. Various providers have their own custom methods of generating this information. For example, in Facebook, you will need to create an Application, where you get this information.
Scope	A comma separated string of the permissions that are required to be granted to the access token. The applicable permission strings are available in API documentation of the respective provider.
Test	Shows the current value of the Field. In this case it's just the generated access token. Note that changing the field attributes does not automatically change the behavior or grant of the token – you need to re-generate the token once you have made changes, using the 'Refresh Token' button.

3.7 Functions

Functions perform a specific action on some input parameters and return a value. Functions can take any number of parameters. You can relate Functions to excel worksheet functions. You can pass static values, Fields or other Functions as parameter to any given Function.

Functions can be embedded within expressions – that's the purpose of its existence. It greatly enhances functionality and flexibility of Expressions.

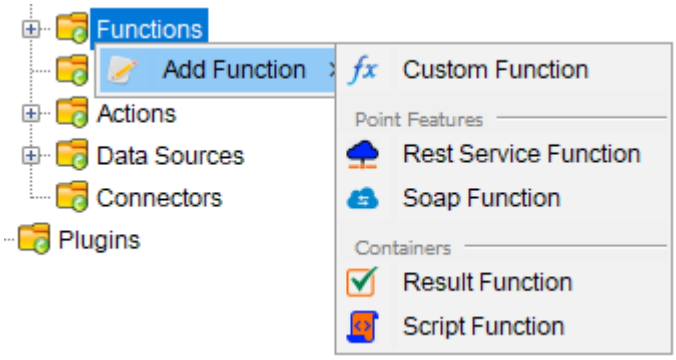


Figure 65 – Add Function menu

3.7.1 Built-in Functions

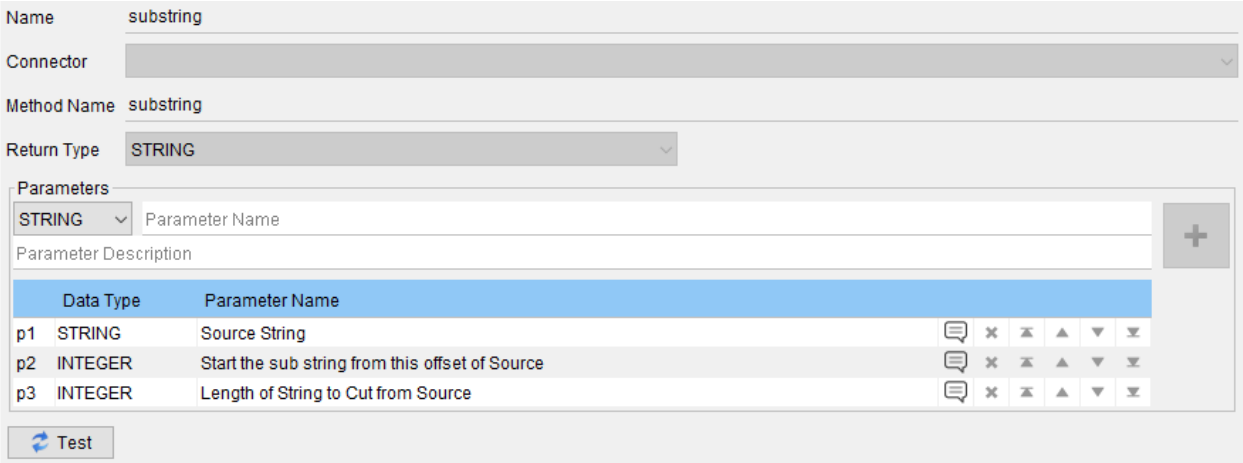


Figure 66 – Built-in Function viewer

Built-in Functions are pre-defined helper functions that are bundled with PARE. It includes common utilities like:
(**Note**: This list is subject to change in future as additional functions may be added)

char	Converts a decimal number to corresponding UTF-8 character. This is useful to embed special chars (like new-line) in the expression.
substring	Cuts a string. Parameters: P1 – Input string P2 – Start Offset (0 based index) P3 – Length of the cut (0 if cut till end)
strlen	Returns length of string. Parameters: P1 – Input string
indexof	Returns the index of the 'n'th occurrence of the specified delimiter (sub-string). Parameters: P1 – Input String P2 – Delimiter

	P3 – Number of delimiters (n)
replace	Replace a char sequence with another. Parameters: P1 – Input string P2 – Char sequence to be replaced P3 – Char sequence to replace with
containsString	Returns True if a substring is present, else returns False. Parameters: P1 – Input string P2 – Substring to find
trim	Removes starting and ending spaces from a string. Parameters: P1 – Input string
timestamp	Returns the current time in dd-MMM-yyyy HH:mm:ss format. Parameters: None
uuid	Returns a unique string. Parameters: None
tolower	Coverts all uppercase chars in a String to lowercase. Parameters: P1 – Input string
toupper	Coverts all lowercase chars in a String to uppercase. Parameters: P1 – Input string
removeextraspace	Trims all occurrences of multiple consecutive space characters in the input string into a single space. Parameters: P1 – Input string
filetotext	Reads a file's content as expression, parses it and returns the evaluated content as String. Parameters: P1 – File path
filetobase64	Converts a file (binary files included) to base64 format and returns the same as String. Parameters: P1 – file path
base64tostring	Performs a base64 decode operation (converts from base64 to UTF-8 encoded String). Parameters: P1 – base64 encoded String
getLastError	Returns the last error message that was encountered. Parameters: None
getLastErrorStack	Returns the last error stack trace
getParam	Runs the parameter parser and returns the 'n'th argument. Parameters: P1 – String containing comma separated values P2 – index of the parameter
gettoken	Runs the tokenizer parser and returns the 'n'th token. Parameters: P1 – String containing delimiter separated values P2 – Delimiter string P3 – index of the parameter
parseJson	Parsees JSON strings. Parameters: P1 – Input JSON string P2 – Key
onError	Changes output on encountering error in parsing expression. Parameters: P1 – Input expression (delegated: use double quotes) P2 – On error replacement expression
onNull	Changes output on encountering NULL value while parsing an expression. Parameters: P1 – Input expression (delegated: use double quotes)

	P2 – On error replacement expression
patchError	Changes the error text while evaluating any expression. Parameters: P1 – Input expression (delegated: use double quotes) P2 – Error message to be returned in case any error is encountered
encrypt	Encrypt a string using a built-in cipher key. Parameters: P1 – String to be encrypted
decrypt	Decrypt a string using a built-in cipher key. Parameters: P1 – String to be decrypted
escapexml	Prepares a String for insertion into XML – when the XML is being prepared manually. This is not required when PARE Models are used to automatically create XML. Parameters: P1 – String to be excaped
command_out	Runs an OS command and returns the output captured from the standard output stream of the command shell. Parameters: P1 – Command string (including parameters)
urlencode	Performs url-encoding on the input string and returns result. Helps in constructing GET URLs which requires url-encoded parameters. Parameters: P1 – String to be url-encoded
urldecode	Performs url-decoding on the input string and returns result. Helps in retrieving data from URL encoded strings. Parameters: P1 – String to be decoded
checkif	Implements conditional return value. Example usage: {checkif([usage] > 200, "[premium_category]", "[normal_category]")} Parameters: P1 – Condition expression P2 – Expression to be used if P1 is TRUE (delegated: use double quotes) P3 – Expression to be used in P1 is FALSE (delegated: use double quotes)
compute	Invokes the PARE math parser on the expression in parameter. Returns a DOUBLE value. Example: {compute([call_count]/[day_count]*30)} Parameters: P1 – Formula expression
eval_condition	Invokes the PARE comparison parser and returns a BOOLEAN (true or false) result. Example: {eval_condition([user_age] < [senior_citizen_age])} Parameters: P1 – Condition expression
pdftext	Returns all textual content as a String from a PDF document. This routine helps search within one or more PDF documents. Parameters: P1 – PDF file path
contextHttpHeader	In Server mode, this function allows retrieval of value of a specific HTTP header from the HTTP request.

	Parameters: P1 – Name of the Header
contextHttpUrl	Returns the full request URL that has been used to invoke a service by hosted in PARE Server. This method allows inspection of the URL. No parameters.
contextUsername	Returns the username of the context API call request. This function works on in Server API call context.
contextUserAttributes	Returns the Attributes of the user of the context API call request, in JSON format. This function works on in Server API call context.
contextUserAttributeValue	Returns the value of an attribute of the context user. Parameters: P1 – Name of the Attribute
contextPartition	Returns the context partition when an API call is made. For public API, the partition may come from the URL path. For protected calls, it comes from the context user.
contextAccessToken	Returns the access token of the context API call request. This function works on in Server API call context.
getWorkflowData	Return the value of a data field in the Workflow instance in context. This built-in works in the context of an executing workflow instance, including Automatic Tasks, Finish Checks and similar. Parameters: P1 – Name of the data field
workflowInstanceId	Returns the Instance ID of the workflow instance in context. This built-in works in the context of an executing workflow instance, including Automatic Tasks, Finish Checks and similar.
workflowParentInstanceId	Returns the Parent Instance ID of the workflow instance in context. This built-in works in the context of an executing workflow instance, including Automatic Tasks, Finish Checks and similar.
workflowInstanceState	Returns the state of a workflow instance. Parameters: P1 – Instance ID
workflowInstanceCreatedBy	Returns the username of the instance creator – if any. Parameters: P1 – Instance ID
dmsDownloadBackend	Downloads a file from server into the specified folder. Parameters: P1 – fileuid P2 – folder path
dmsUploadBackend	Uploads a file into DMS and returns the fileuid. Parameters: P1 – file path
dmsDownloadFileData	Downloads the file in a FileData JSON format. FileData format includes: { filename: "…" content: "<base64 encoded gzipped file content>" contentEmbedded: "true"

	} Parameters: P1 – fileuid
dmsUploadFileData	Uploads a File Data JSON into DMS. Parameters: P1 – File Data JSON with contentEmbedded = true

fileInfo	Returns information about a DMS file entry in following format:
	
	{
	"partition": "default",
	"engine": "monitor",
	"filename": "<redacted>",
	"fileuid": "155exxxx-xxxx-xxxx-xxxx-xxxxxxxx0a0f",
	"attributes": "{}",
	"createdby": "admin",
	"public": "true",
	"modonts": "24-Jul-2023 23:06:36",
	"url":
	"http://localhost:8080/RuleServer/files/default/monitor/155exxxx-xxxx-xxxx-xxxx-xxxxxxxx0a0f/stream"
	}
	
	Parameters:
	P1 – fileuid

3.7.2 Custom Functions

Custom functions are 'methods' belonging to a particular object exposed in PARE. Custom Functions share the same editor screen as Built-in Functions.

Custom Functions have the same internal format as Built-in Functions. In this case though, user needs to attach a pre-defined Class Connector to it. The object is scanned to find a method which matches the signature of the Function and that is invoked every time the user function is called at the time of expression resolution.

The class loader can automatically scan the location (full sub-directory structure from a given location) including JAR files and auto-populate the list of loadable Classes.

Class Connector	The Class Connector node that is going to provide the Class template and object instance.
Set	Set button actually sets the object instance and the method signature check is done at that point.
Method Name	The Class Method that is called when the User Function is invoked
Return Type	The PARE DATATYPE of the value being returned by the User Function
Parameter Type	PARE DATATYPE of the parameter being passed to the function
Parameter Name	Name of the parameter Note: Multiple parameters can be attached to a User Function
Test Expression	Expression Field that is evaluated on clicking Button Text. Ctrl-space brings up available options. Functions will follow regular expression syntax of {function_name(parameters)}.
Test	The Test Expression is evaluated and shown on screen.

Note:

- As a best practice, users should create User Functions and Custom Actions via the right-click menu option of a Class Connector. This helps in implementing correct naming conventions.
- If Java 8 is used to compile the Custom function JARs, the appropriate compiler options should be selected so that PARE can use the actual parameter names in Java code to create PARE Function parameters. For example, in Eclipse, the following options should be selected:

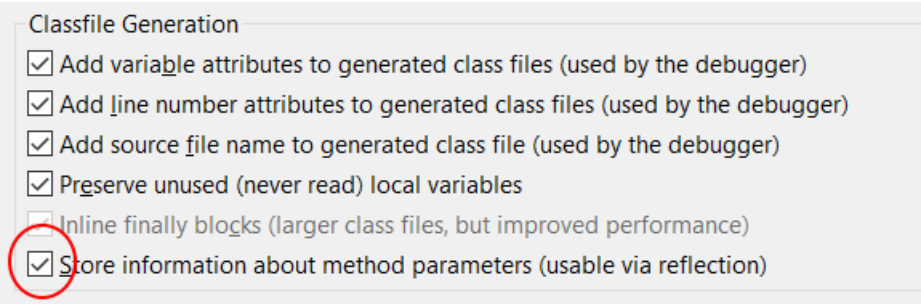


Figure 67 – Recommended Eclipse Java compiler options

3.7.3 Statistical Functions

For DataSource fields, user can create a set of Statistics Functions that return the following:

- Count – number of corresponding fields in the Data Source
- Minimum – Minimum value (if field is numeric)
- Maximum – Maximum value (if field is numeric)
- Sum – Sum of all values (if field is numeric)
- Mean – Average value (if field is numeric)
- Median – Media value of the field
- Variance – Statistical variance
- StDev – Standard Deviation of the distribution

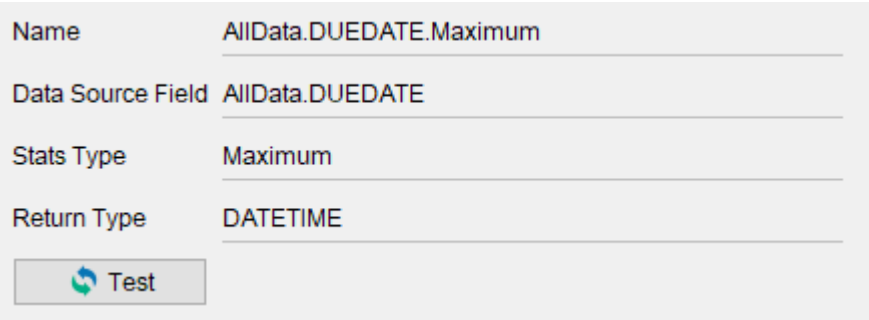


Figure 68 – Stats Function

Name	Name of the Function
Data Source Field	Underlying data source field that provides the Data for stats calculation.
Stats Type	Any one of Average, Sum, Max, Min or Count.
Return Type	If type equals Count, return type is LONG. For the rest, return type is DOUBLE.
Test	Shows the applicable Stats value. Note that this will typically return 0 (zero), unless the Stats Refresh Action has been already executed.

Note:

- The Stats functions return a valid value when the statistics data is collected for the DataSource. Stats data gathering is done via a separate Action that can be created for the corresponding DataSource.
- Stats Functions are created by right-clicking the DataSource Field and selected appropriate menu option. As below:

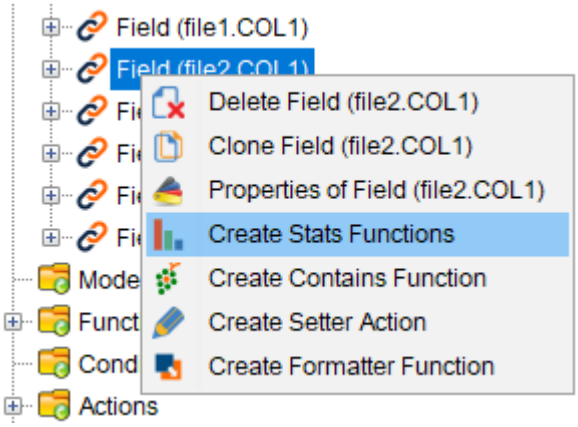


Figure 69 – Create Stats Functions

On clicking, the following popup lets user select which Stats function will be created:

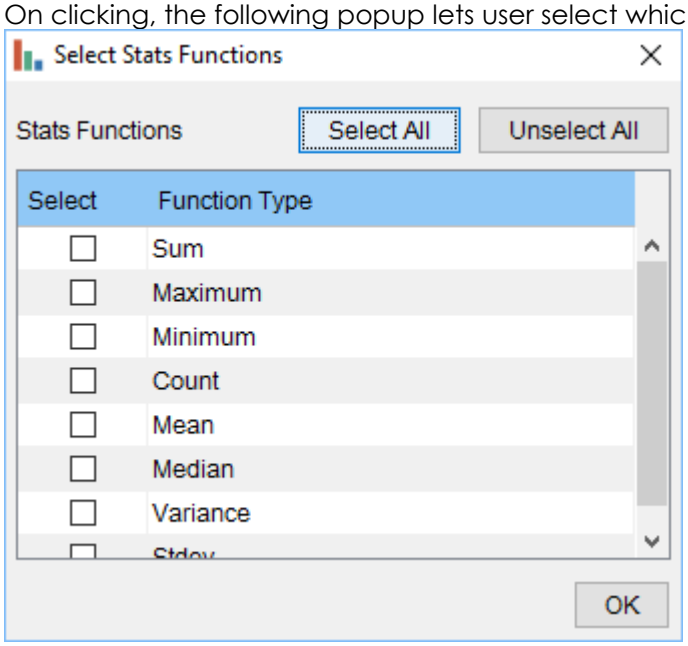


Figure 70 - Select Stats Functions to create

3.7.4 Contains Function

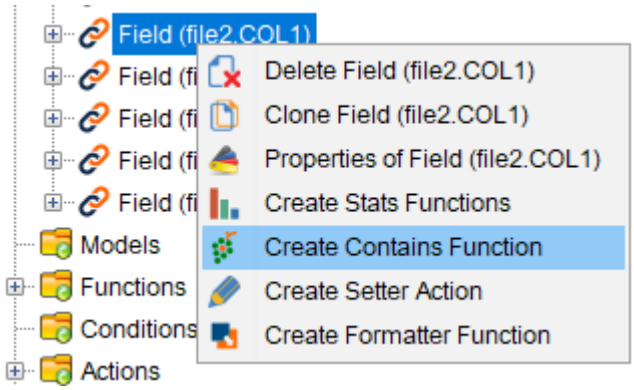


Figure 71 - Create Contains function

'Contains' function is associated with a Data Source field and checks if the passed parameter exists among all the possible values of the field – i.e. all rows of the Data Source. Please use this carefully as it triggers a full Data Source scan. If the Data Source has many rows – say a large DB table – then this may take a long time to finish.

3.7.5 Socket Read Function

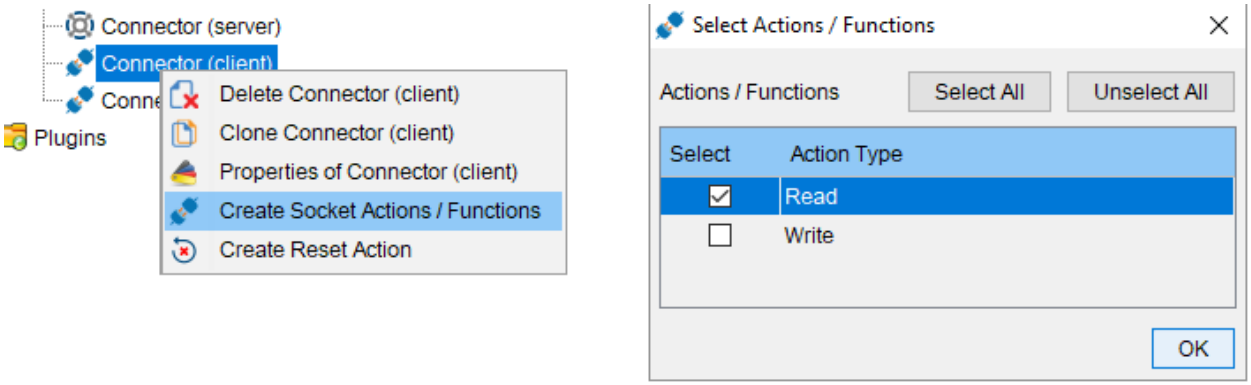


Figure 72 - Create Socket Read Function

The Socket Read function reads from a TCP Socket or a UDP Socket and returns the data as String. In case the read length parameter value is > 0, the call blocks until required data is available. If the read length parameter is < 1, the call blocks until at least 1 byte of data is available in the Socket.

Socket Read Function depends on the Client Socket connector.

3.7.6 Server Socket Read Function

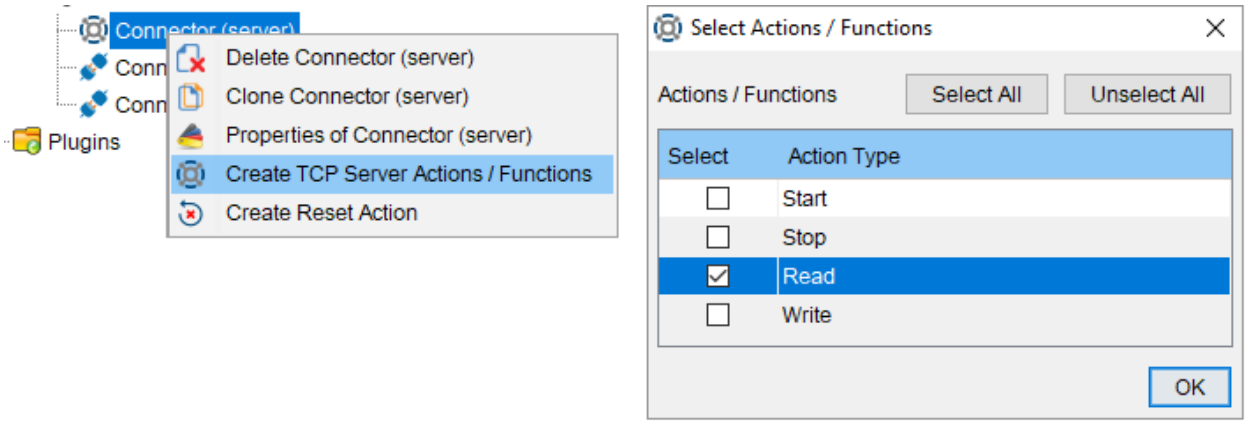


Figure 73 - Create TCP Server Socket Read function

The Server Socket Read function is similar to Socket Read, but it works on a connected socket when a remote client connects to the built-in TCP server in PARE. It depends on the Server Socket Connector.

3.7.7 Result Function

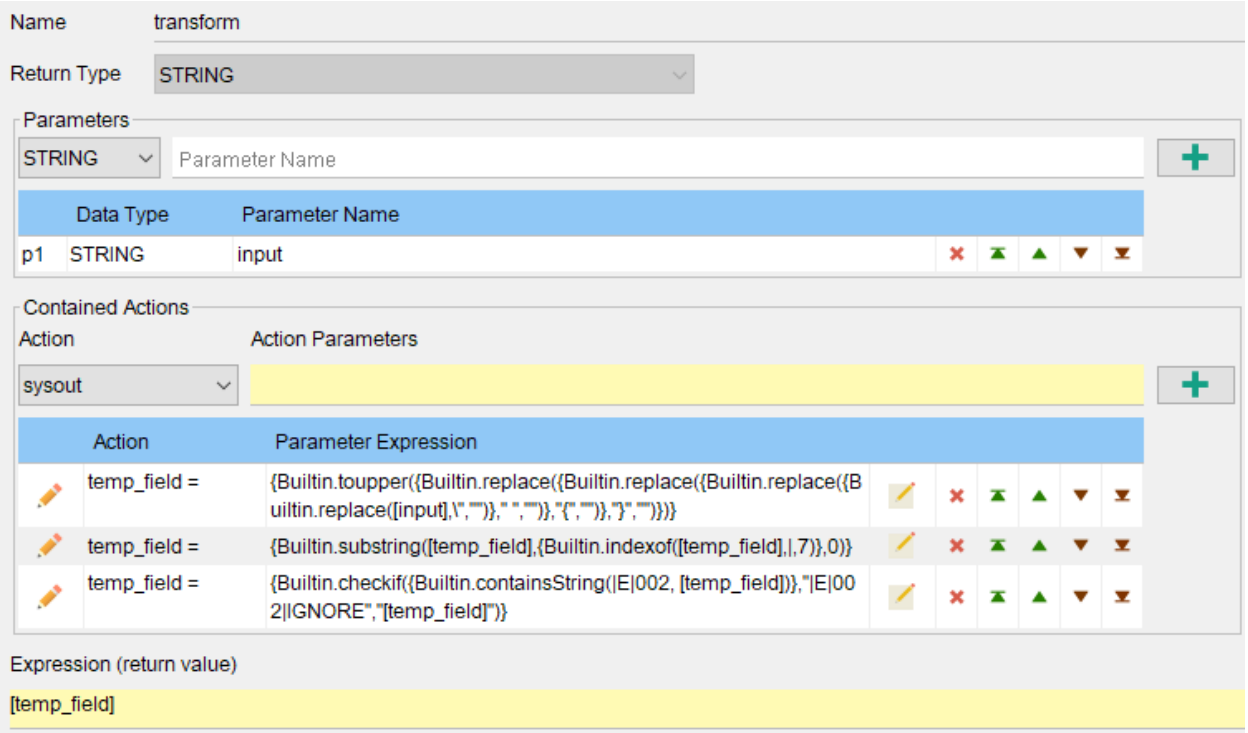


Figure 74 - Result Function editor

Result Functions combine Actions and a returned Value. While all other functions just return pre-configured dynamic value, this Function can combine Actions and then return the resulting value in one go.

Name	Name of the Function
Return Type	Data type returned by the Function (STRING, INTEGER, …)
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Actions	List of Actions (Action name / action parameters) to be executed as part of executing this Function.
Expression (return value)	Expression that sets the value of the field. Arithmetic parsing is done on the expression if the return type is numeric.
Test Expression	Expression Field that is evaluated on clicking Button Text. Ctrl-space brings up available options. Functions will follow regular expression syntax of {function_name(parameters)}.
Test	The Test Expression is evaluated and shown on screen.

Note: It's possible to implement same functionality by making a series of calls from the client. This pre-configured step makes for lesser coding at client end.

3.7.8 Formatter Functions

Formatter Functions are meant for Date and Numeric output formatting. By default the output format is as below:

DATE: dd-MMM-yyyy
TIME: HH:mm:ss
DATETIME: dd-MMM-yyyy HH:mm:ss
INTEGER: #
LONG: #

FLOAT: #.########
DOUBLE: #.########

However, if the output is desired in any other format, the Formatter function should be used. Formatter functions are created from right-click menu option of Date type Fields as below:

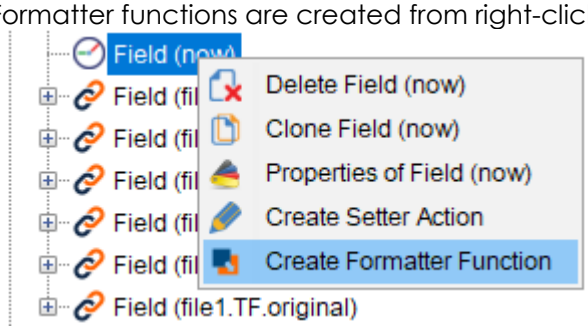


Figure 75 - Create Formatter Function

Example usage: "insert into `time_table` values ('{now.Formatter(yyyy-MM-dd HH:mm:ss)}')"

Note: For date/time formatters, a special keyword 'epoch' can be used to retrieve the number of milliseconds since 01-Jan-1970 00:00:00 GMT. This is useful when you need to do date comparisons as condition expressions, for example: {adate.formatter(epoch)} > {bdate.formatter(epoch)}.

3.7.9 Model Json / XML Functions

This Function returns the JSON or XML string from an existing Model.

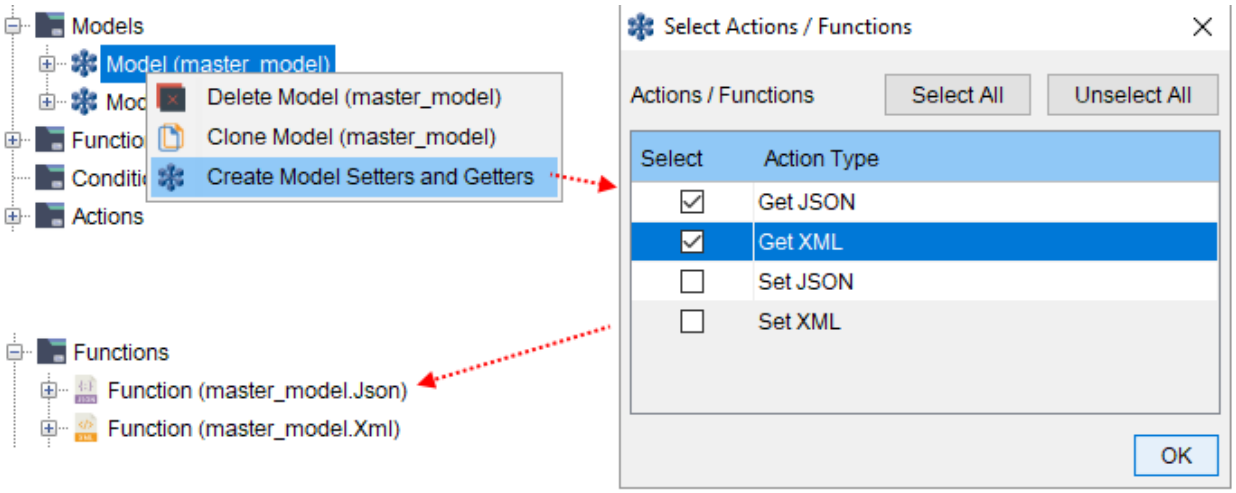


Figure 76 - Create Model JSON / XML Functions

These functions implement Model Serialization in either XML or JSON format. These functions are created via the corresponding Model right-click menu options. The naming conventions of the Functions are <Model name>.Xml and <Model name>.Json.

Note: The corresponding Model must be successfully deployed before these functions can work.

3.7.10 Rest Service functions

REST Service function implements a configurable method to call REST services and return the result. It uses the OKHTTP client and uses PARE Parameters to map the different service inputs. As any other PARE Function, it can be embedded within expressions.

The UI allows setting of the base URL, Authorization, HTTP method, headers, body content and proxy configurations.

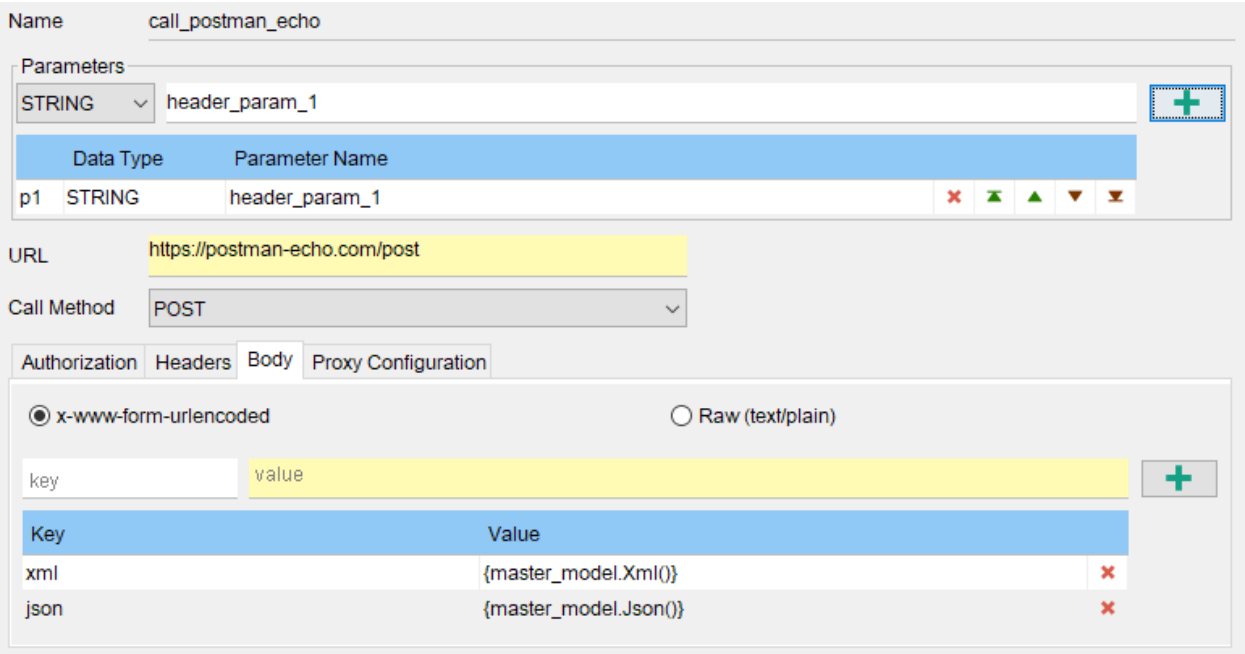


Figure 77 - Rest Service function editor

Authorization:

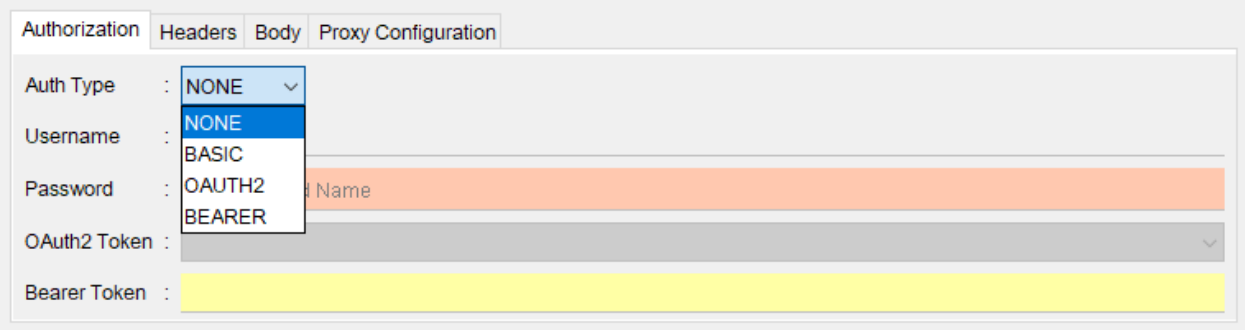


Figure 78 - Authorization options

- NONE: No Authorization header
- BASIC: Basic Authorization header included (username and password)
- OAUTH2: An OAUTH2 Bearer Token Authorization header included (uses an Oauth2Token Field)
- BEARER: A Bearer Token Authorization header included (uses PARE expression to resolve Token)

Headers:

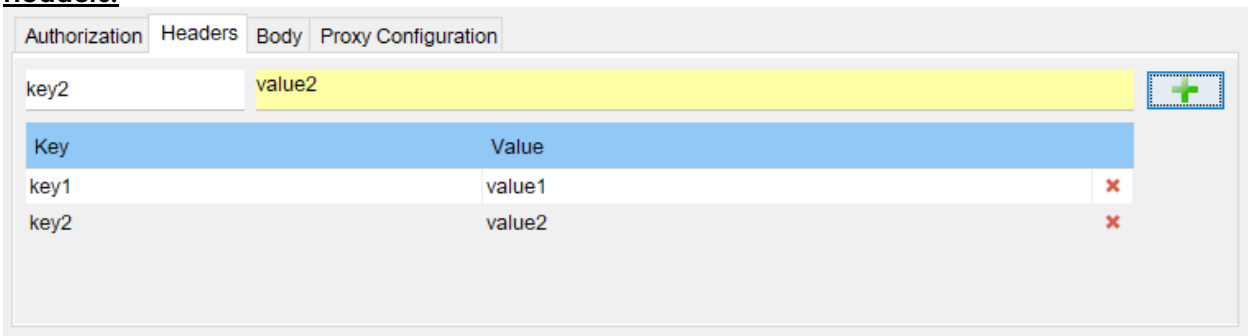


Figure 79 – HTTP Header options

Insert custom headers in the HTTP request as required. All values are PARE expressions.

Body:

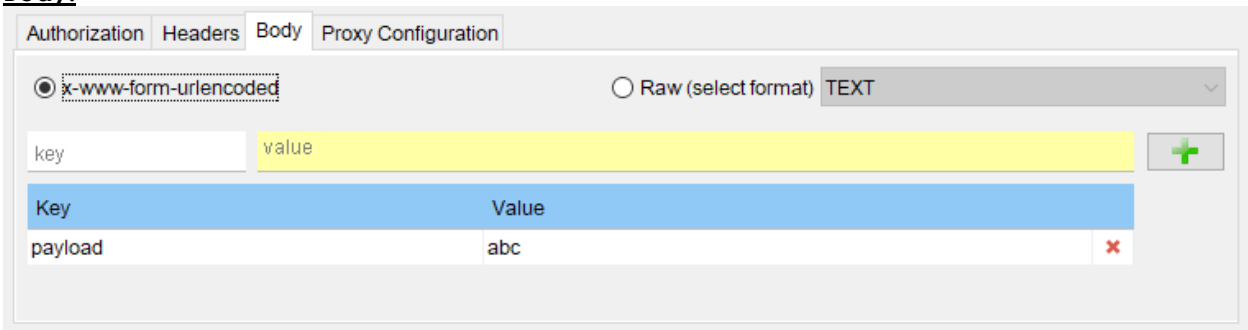


Figure 80 - Body content - form type

Use this option to pass Form data (in the form of a list of Key-Value pairs) as part of the Request Body. Note that only POST, PUT and DELETE type requests can carry such Forms.

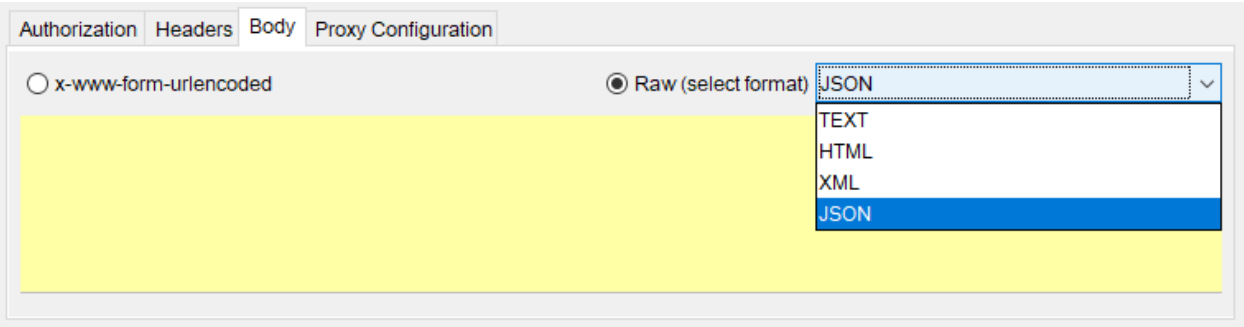


Figure 81 - Body content - raw type

Use this option to pass custom data as body of the call. Any PARE expression can be used and user can specify the type of data being passed – ex: text/plain, text/html, application/xml or application/json.

Proxy:

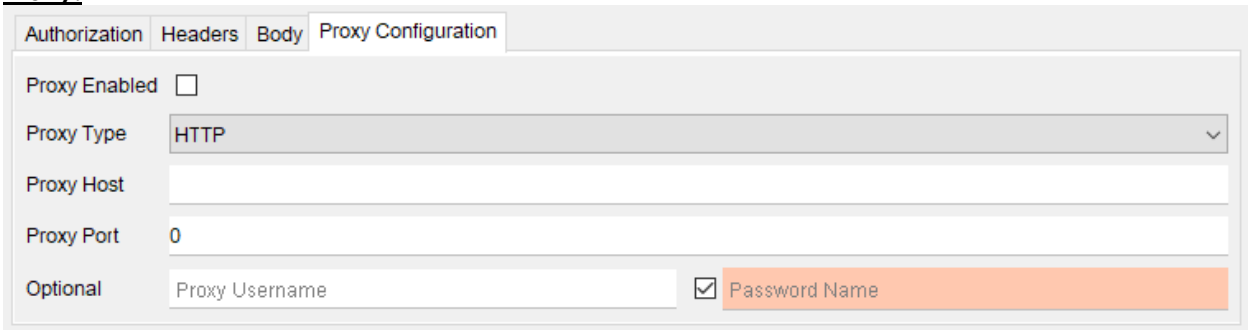


Figure 82 - Proxy configuration

Use this to configure Proxy options, in case there is no direct connection to the target URL. Note that username and password credentials are not currently supported.

Remaining options:

Name	Name of the Function
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
URL	Base URL of the REST service. This should not include the parameters (in case of GET method).
Call Method	GET, POST, PUT or DELETE (mechanism used to pass data to the service)
Test	The Test Expression is evaluated and shown on screen.

3.7.11 Script Function

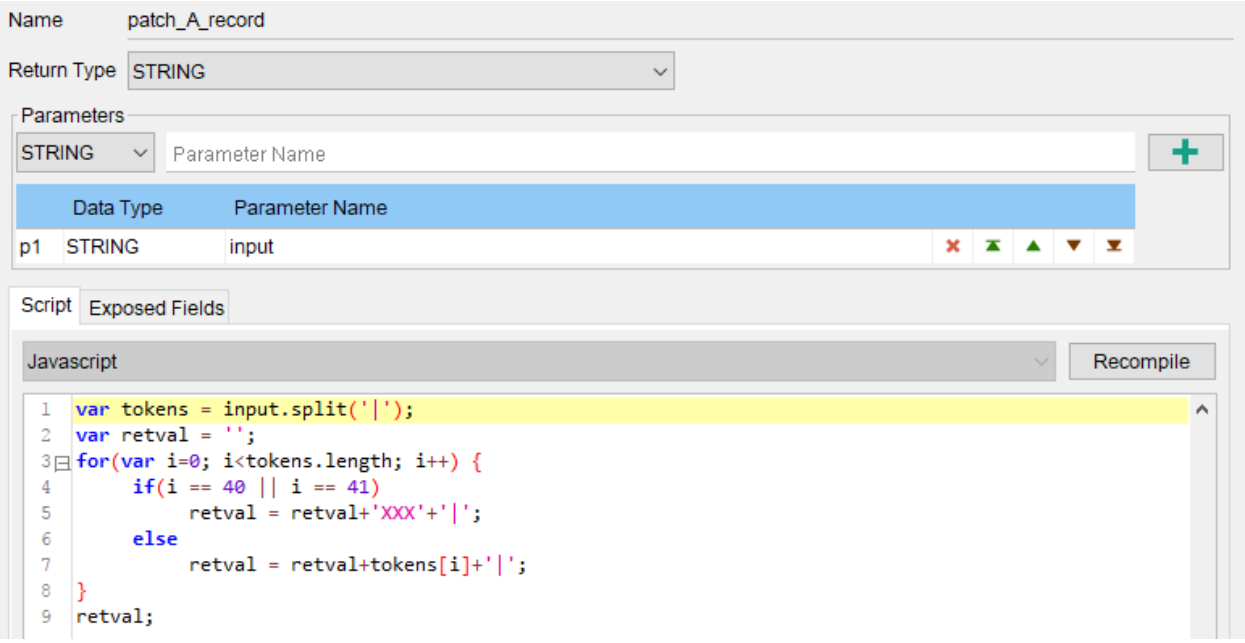


Figure 83 - Script Function editor

Script Functions enable embedding of Scripts within PARE configurations. This is a powerful way of configuring complex algorithms within PARE.

Name	Name of the Function
Return Type	Return DATATYPE of the Function
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Select Field	Select Fields one by one to expose in the Script
Select All	Select ALL PARE fields to expose in the Script
Exposed Fields	PARE Fields exposed (copied) within the Script
Script Type	JavaScript or LUA (other languages may be added in future)
Script Editor	Syntax aware editor for writing the Scripts.
Test	The Test Expression is evaluated and shown on screen.

3.7.12 BAPI Function

BAPI Functions are generated from an existing SAP Connector. It's used to directly call SAP ABAP routines using BAPI and return the data in XML format. The return value can then be attached to a XML data source and mapped to fields as required.

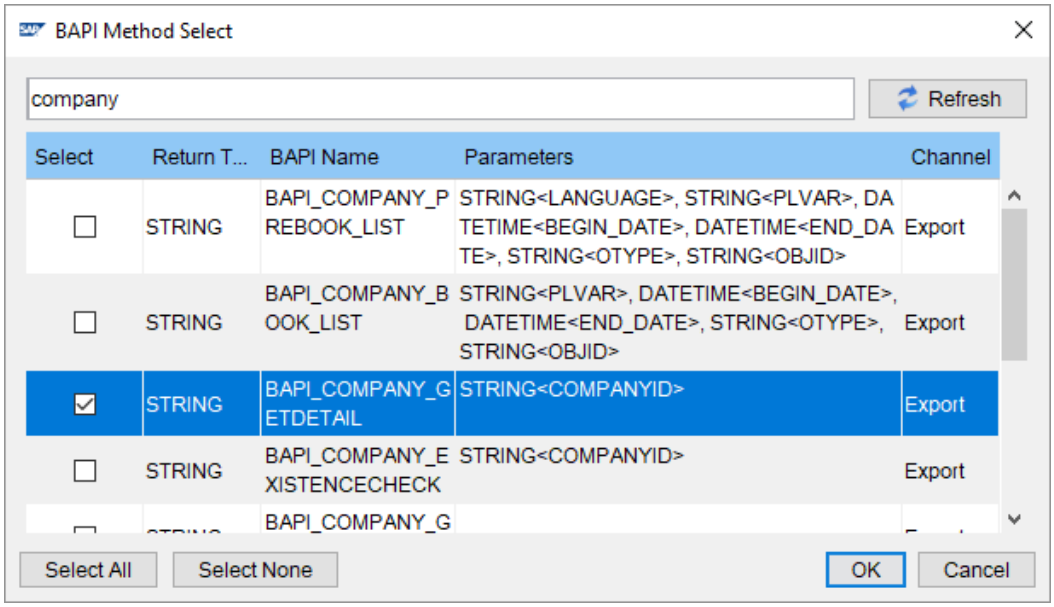


Figure 84 - BAPI Method selector

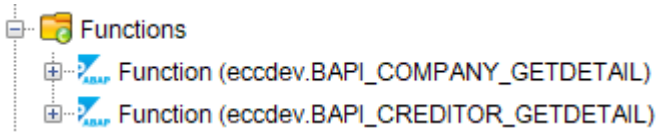


Figure 85 - BAPI Methods in tree browser

Only those BAPI methods can be selected which take input parameters of one of the PARE supported data types. Other methods do not appear in this list. Providing a filtration criterion ('company' in the above image) is strongly recommended so that the search does not take too long.

Users should also select the right channel ('Export' is default) through which the return data is provided. We strongly recommend consulting your SAP resource before starting to use this feature. The return data is always in XML format. The same should be used to configure a XML Data Source and then access the data through Data Source fields configured over the XML DS.

Note: to use SAP connectivity, users will need to put 'sapjco3.jar' in classpath. SAP Customers should be able to download this library and sapjco3.dll from their Service Marketplace. Since the DLL file is dynamically located, its recommended to store the DLL in the same folder as sapjco3.jar.

3.7.13 SOAP Function

SOAP Function defines an actual SOAP API call. PARE provides facility to browse the WSDL (attached to the SOAP Connector), select the API to call and automatically extract the XML envelops.

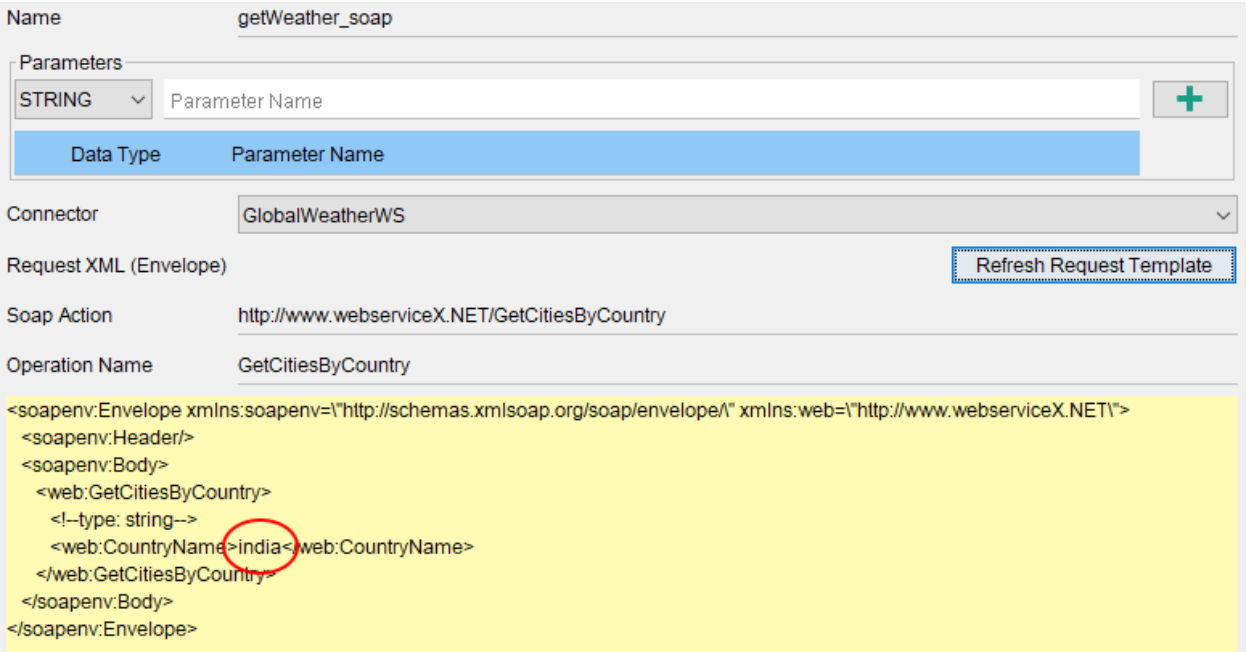
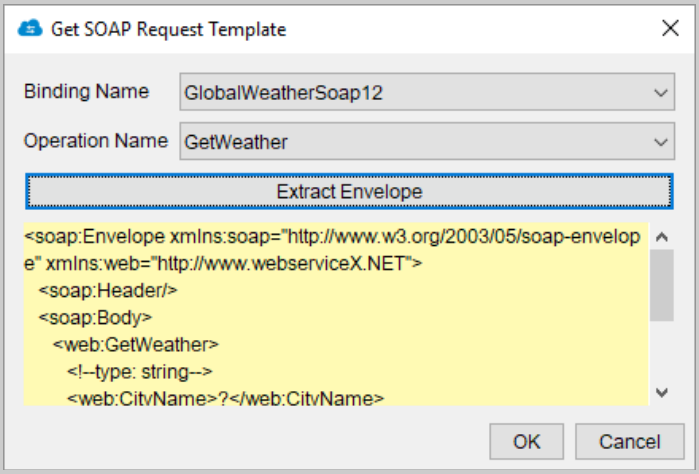


Figure 86 - Soap Function editor

Note: SOAP Function always returns an XML. Users may choose to either use {xmlpath} builtin function to extract data from the XML or configure an XML Datasource to do more detailed processing of the output.

Name	Name of the SOAP Function
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Connector	The SOAP Connector that will support this Function
Request XML (Envelop)	This area captures the API details: • Soap Action • Operation Name • Request envelop All of them are auto-generated from the WSDL. Users should invoke the operation chooser by clicking the 'Refresh Request Template' button. This brings up a dialog like below:  **Figure 87 - SOAP Operation chooser** Once the operation is chosen, the XML envelop is copied in the Function,

	where the input fields are marked with a '?'. Users are now free to embed actual values or PARE expressions in the envelop.
Test	The Test Expression is evaluated and shown on screen.

3.7.14 Case Function

Case Function allows return a specific value based on a Case Slot. This is similar to Case Table Action, only a Value is returned per Slot instead of calling a series of Actions.

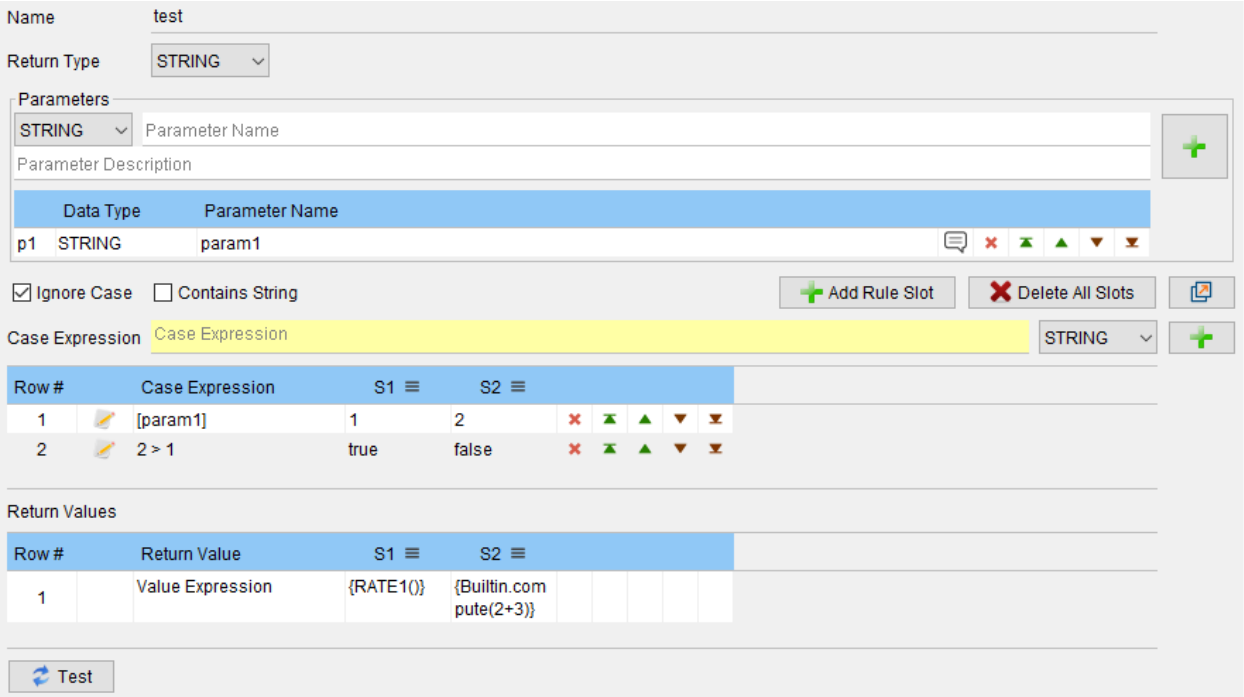


Figure 88 - Care Function Panel

This is a variant of the Case Table Action where instead of random Actions per slot, there is a single return value per slot. Case Function (CF) configurations are compact and suitable for specific use cases like Rating Rules.

The remaining characteristics and behavior are like Case Tables and hence may be referred to in the Case Table section.

Name	Name of the Function
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Ignore Case	Ignore case when comparing values with the case expression values
Contains String	If checked, the comparison method checks if the Slot value exists within the case expression. Otherwise, the slot value should be equal to the case expression value.
Add Expression	Add a Value expression to the CT. Note that for each Rule Slot, the actual value of the expression will have to be entered in the Case Table. This can be done directly by entering value in the table cell.
Add Rule Slot	Adds a Slot in both the Cases and Actions tables.
Delete all slots	CT can be complex, and if you feel it's all messed up, use the 'Delete All Slots' button to nuke the configuration and start from scratch.

Configuring the Slots	User needs to enter the right combination of Cases (value comparison) and Return Value Expression per Slot. Each of the values are checked in their separate slot. It's common to have more than one Value configured in a single slot. **Note**: When the CF is executed, each slot is executed one by one, and in sequence until a value is returned. Only one value is returned from a CF.
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

3.8 Conditions

Conditions return either TRUE or FALSE. Conditions include one or more conditional operators or one or more other Conditions. Conditions coupled with exposed Data allow definition of most business decision points.

Conditions can be of 3 types:

- Simple – compares an Existing Field with an expression
- Expression – supports a comparator expression
- Joined – joints other Conditions

Following Conditional operators are supported (for Simple and Expression Conditions):
EQ (==), NE (!=), GT (>), GTE (>=), LT (<), LTE (<=)

Following Join operators are supported (for Joined Conditions):
AND, OR

Conditions are created by Right clicking on the Conditions Tree node and selecting the appropriate type:

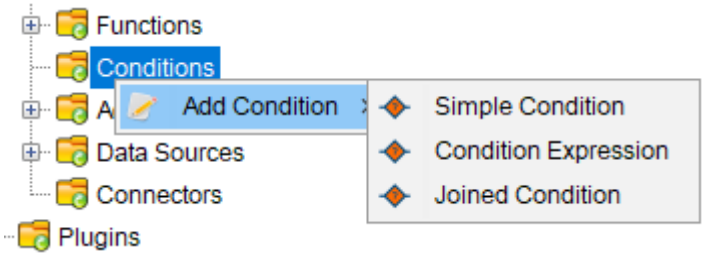


Figure 89 – Add Condition menu

3.8.1 Simple Condition

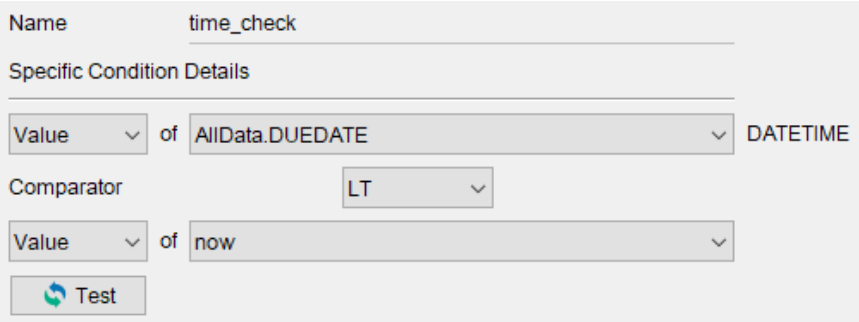


Figure 90 – Simple Condition editor

Simple conditions are the most flexible, as it can handle Date and Time type of Fields, supports optional fields and IN condition. But it is restricted to only one conditional operator.

Name	Name of the Condition

Stats Type 1	Value/Sum/Average etc of
Field 1	First field to be compared with
Comparator	Any one of: EQ, NE, GT, GTE, LT, LTE, IN. The last one, IN, requires that Field 2 is a Data Source Field.
Stats Type 2	Value/Sum/Average etc of
Field 2	Second field to be compared to
Test	Performs evaluation and shows result

Note:

- If a field can be set as 'non-mandatory' and used in a Simple Condition, the result of the Condition will always be TRUE.
- If the IN clause is used, the 2nd field (Field 2) must be a Data Source backed Field.
- The first field (Field 1) determines the DATATYPE used for comparison. So, for example, if Field 1 is a DOUBLE and Field 2 is a STRING, an attempt will be made to convert the STRING to its equivalent DOUBLE value before comparison.

3.8.2 Expression Condition

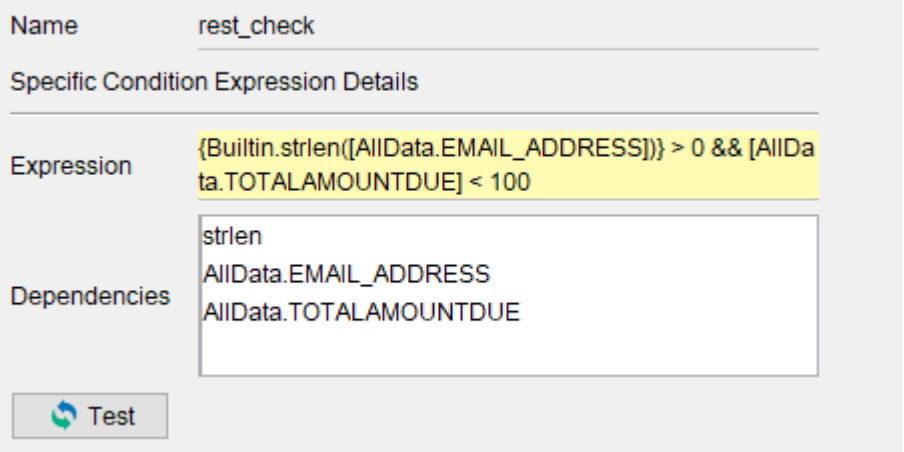


Figure 91 – Expression Condition editor

The syntax of Condition Expression is like Java or JavaScript. The syntax is well known, supports brackets, can include multiple comparisons.

Name	Name of the Condition
Expression	Evaluation expression that is parsed by the Condition parser and can process following conditional operators: <, <=, >, >=, !=, ==
Dependencies	Shows other fields or functions that are referred in the expressions used in this Field.
Test	Performs evaluation and shows result

Note: As of current release the '!' symbol is un-supported. Which means, expressions like "![val2]" cannot be evaluated. If val2 is Boolean, use the expression "[val2] != true" instead.

3.8.3 Joined Condition

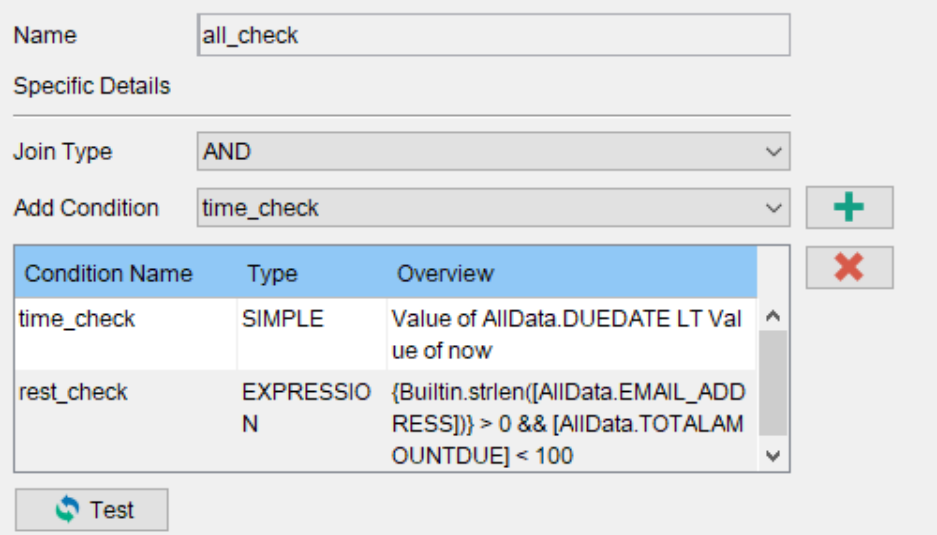


Figure 92 – Joined Condition editor

A Joined Condition joins multiple other Conditions with either AND or OR.

Name	Name of the Condition
Join Type	Following Join types: AND, OR can be used. Note: A single Joined Condition can use either AND or OR, not both.
Add Condition	Child conditions that can be joined
Test	Performs evaluation and shows result

Let us consider the following real-life condition: "Customer is registered more than 5 years ago and either current bill amount is greater than 500 or customer is a senior citizen."

This needs 3 simple conditions as follows:
1 – SCOND1: [diff years] >= 5
2 – SCOND2: [bill amount] > 500
3 – SCOND3: [customer age] > 60

Note:
- [diff years] can be DateDiff field: [relationship start date] – [today]
- [customer age] can be DateDiff field: [customer DoB] – [today]
- [bill amount], [relationship start date] and [customer DoB] may be DB Fields mapped to Customer record in DB

Now we can join the Rules as follows:
1 – JCOND1: SCOND2 OR SCOND3
2 – **FINAL CONDITION**: SCOND1 AND JCOND1

3.9 Actions

Actions perform business functions. Actions based on Conditions are Rules. PARE supports different types of Actions and provides Container type Actions that can include other Actions. The supported Action types are:

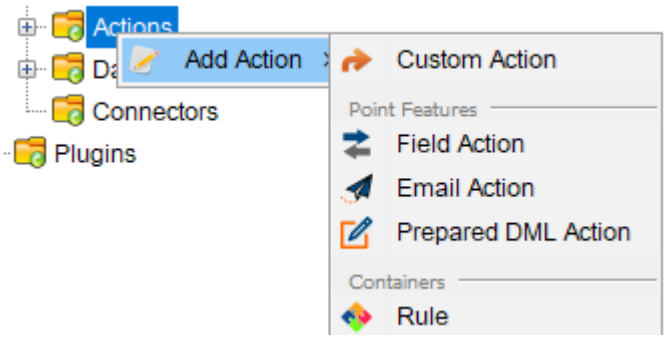


Figure 93 – Add Action menu

3.9.1 Common characteristics of Actions

There are two basic types of Actions, viz:

- Executors (Custom and Point Feature Actions) – These actions perform some specific tasks, say, set the value of a field, create a File or call a web service. Examples of this type of Actions are any Custom Action, Command Action, Field Action, DML Action, and all the Built-in Actions.

- Containers – These actions bundle other Actions and form the basis of creating higher level application flows. Examples of this type of Actions are Rules, Decision Tables, Jobs, Sequences, On-Error Actions and Scheduled Actions.

All Actions can take parameters and can pass on the same to child actions or commands via expression. The parameters can be embedded in expressions using following notation:
[param 1 name] - first parameter
[param 2 name] - second parameter
…

Parameters should be used judiciously to preserve readability of the configuration. For example, On-Error Actions can use parameters as they just provide an error handling wrapper to underlying Actions. Parameters should be used to maintain modularity when linking one or more configurations. However, too many parameters may negatively impact readability and may become maintenance overhead.

3.9.2 Built-in Actions

Built-in Functions are pre-defined helper Actions that are bundled with PARE.

This list is subject to change in future as additional actions may be added:

sysout	Writes a String to STDOUT. Parameters: P1 – Input string
break	This action performs the equivalent of a "break" out of a looper. This works on Loop and single-threaded Job Actions.
nullout	Discards the output of the expression passed as parameter. This is required in situations where we need to execute a function / expression but do not need to do further processing with its output.
exception	Raises an Exception (error indicator) with passed string. Parameters: P1 – Input string
playsound	Plays out a sound from a media file path. Parameters: P1 – sound file path (supported formats only)
sleep	Halts for specified number of milliseconds. Parameters: P1 – Input number of milliseconds
increment	Applies atomic ADD operation on a field. Parameters:

	P1 – Name of the field P2 – ADD amount (ex: 1, 2, 3 …) Note: P2 can be a negative number also
command	Executes a command (full command path and parameters should be specified in the expression). Parameters: P1 – Full command string <u>Note</u>: • Commands are launched as separate child processes, and internal threads are launched to wait for its completion. So, if several Command Actions are fired one after the other, those commands may be executed in parallel. If the commands generate any output, the same is automatically redirected to the STDOUT (default output stream) of the calling process, i.e. user application. • Launching commands with user input is not recommended, nor commands with GUI, which tend to wait for inputs rather than complete on its own. This may lead to un-predictable behavior. • Command Actions are generally resource heavy in nature as child process is created per command and background threads are required to drain the default STDOUT and STDERR streams as well as wait for the command to finish execution. Shell scripts are a good example of actual usage of the Command Action. Commands or shell scripts that may take long time to finish are best avoided, or be launched as a 'background' task using a Sequence container. • Shell built-in commands or shell scripts cannot be invoked directly as the shell needs to be launched first. Hence, as in above example, running the windows 'echo' command requires that 'cmd.exe' (the windows default shell) is launched with options to run the 'echo' built-in command and exit.
ldap_auth	Performs Authentication using LDAP protocol: P1 – Server URL (ldap://<server>:<port>/…) P2 – Authentication type (simple / sasl) P3 – Principal Name (username@domain name) P3 – Password
base64tofile	Creates a file from a base64 encoded input string. Parameters: P1 – base64 encoded string P1 – output file path
processEvent	This builtin is used by a PARE Process to report events that go into the process DB log: P1 – Severity (info, error, debug) P2 – Message (the main message) P3 – Remarks (reference information – ex: the stack-trace in case of an error event)
startWorkflowMonitor	Launches the Workflow Monitor thread in the node – if not already started. Please note that the Workflow Monitor thread is launched on-demand for almost all workflow API calls and user needs to make sure that the monitor thread runs in a single server in a cluster

	deployment.
stopWorkflowMonitor	Kills the Workflow Monitor Thread in the node.
setWorkflowData	Sets a data field value at run time within the workflow instance in context. This built-in works in the context of an executing workflow instance, including Automatic Tasks, Finish Checks and similar. Parameters: P1 – key P2 – value
setWorkflowInstanceData	Sets a data field value at run time in the workflow instance (instance ID passed as parameter). Parameters: P1 – Instance ID P2 – key P3 – value
cancelWorkflowTask	Cancels a workflow task by name within the workflow instance in context. This built-in works in the context of an executing workflow instance, including Automatic Tasks, Finish Checks and similar. Parameters: P1 – task name
workflowEvent	Each workflow instance maintains a separate event log. This built-in is used by Automatic Tasks to log events in the workflow in context. This built-in works in the context of an executing workflow instance, including Automatic Tasks, Finish Checks and similar. Parameters: P1 – Severity (INFO, ERROR, FATAL) P2 – Message (the error message or similar) P3 – Remarks associated with the error message
reassignWorkflowTask	Assign a Manual Task in the context workflow instance to a designated user. This built-in works in the context of an executing workflow instance, including Automatic Tasks, Finish Checks and similar. Parameters: P1 – task name P2 – user name P3 – reason

3.9.3 Custom Action

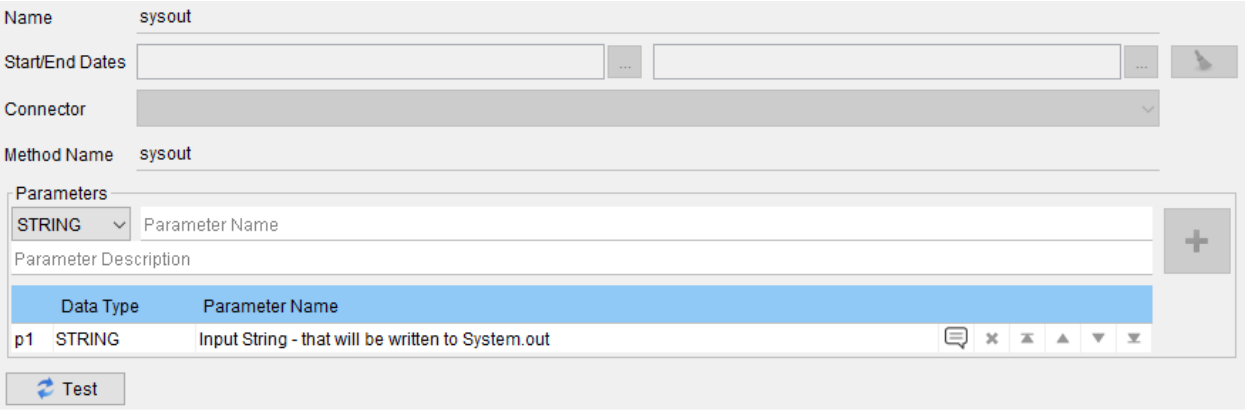


Figure 94 – Custom Action editor

A Custom Action is backed by user code. Template (method signature) creation is done via the PARE UI. Users will either need to attach a Class Connector that actually implements the Action's method.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Method Name	Name of the Java method that is called when the function is executed. Note that for built-in actions, the object exposing the Method is built-in and pre-set.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Action Help	Help String that user may include for better documentation.
Test	Executes the Action, and shows a completion confirmation message after execution of the Action is completed.
Class Connector	The Class Connector node that is going to provide the Class template and object instance.
Set	Set button actually sets the object instance and the method signature check is done at that point.
Load	The Load button invokes the class loader and creates an instance object using the default constructor of the selected class. The method signature check is done at that point.

Note:
1 – As a best practice, users should create Custom Functions and Custom Actions via the right-click menu option of a Class Connector. This helps in implementing correct naming conventions.

3.9.4 Field Action

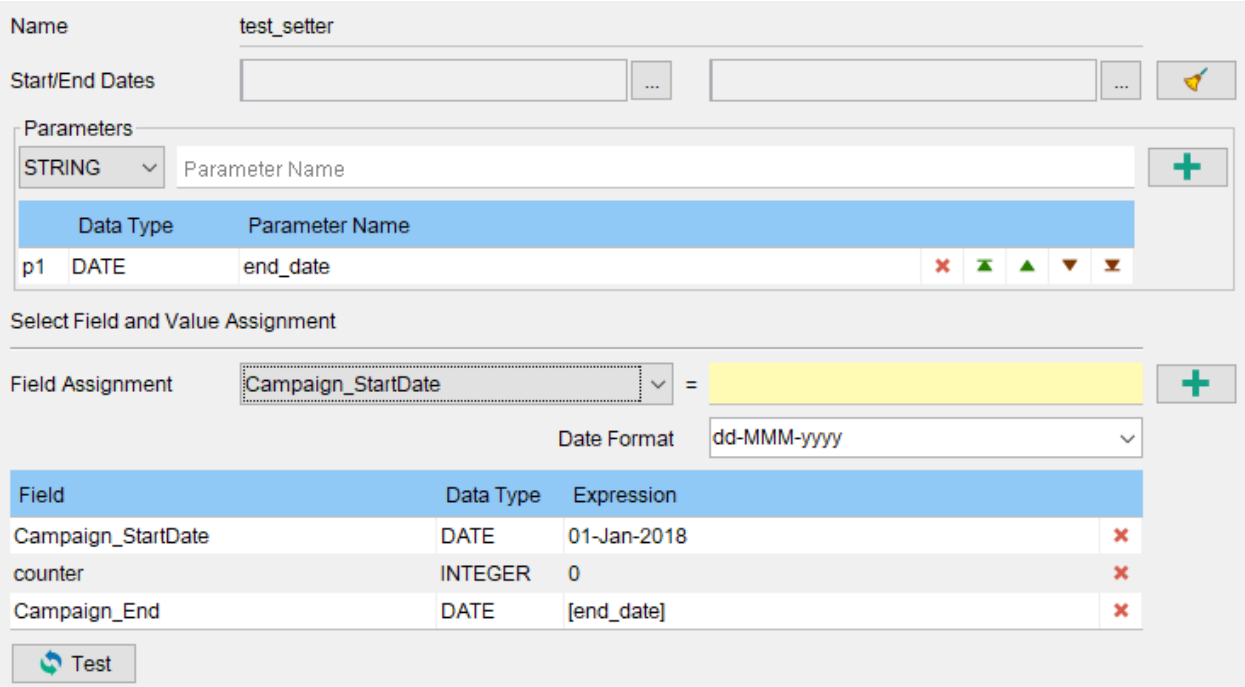


Figure 95 – Field Action editor

Field Actions are used to set value of selected Fields. Multiple Field assignments can be included in this Action. Note: Only writable Fields can be used, as internally the 'setValue' Field API is called. Hence, User, App, and DataSource Fields (currently Object Data Source and Master Data source supports this) can be used in this Action.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Field	A field whose value needs to be set. Note that the field should be of settable type. For example, a User Field's value can be set, whereas a Composed Field's value cannot.
Value expression	Expression that evaluates to value of the Field.
Field Table	List of all Fields and their Value assignments
Test	Executes the Action, and shows a completion confirmation message after execution of the Action is completed.

Note:

- Field Actions should be used when more than one editable Field is being set – for example, updating multiple attributes of a Class Connector object.
- If only one Field needs to be set, it's better to use Field Setter Action as that produces much more readable configuration.
- A suitable scenario of using Field Action is initial configuration of a facility – say setting up an email profile (all the SMTP parameters), or FTP profile.

3.9.5 DML Action

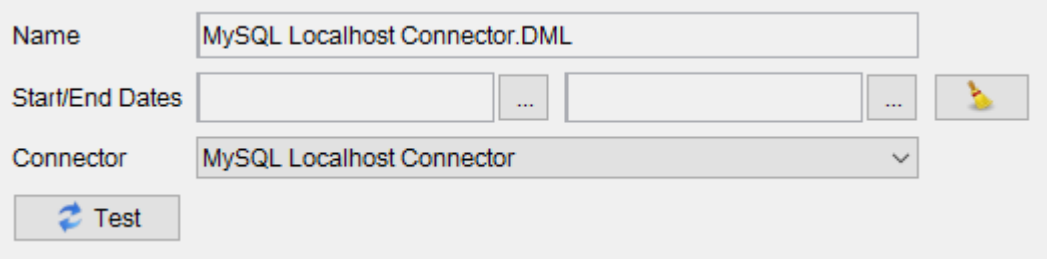


Figure 96 – DML Action editor

DML Actions are used to set values in an RDBMS or change the database in a generic way. Currently auto-commit is turned ON for all such scenarios.

The best way to create a DML Action is to right click on a connector and select the 'Create DML Action' menu option.

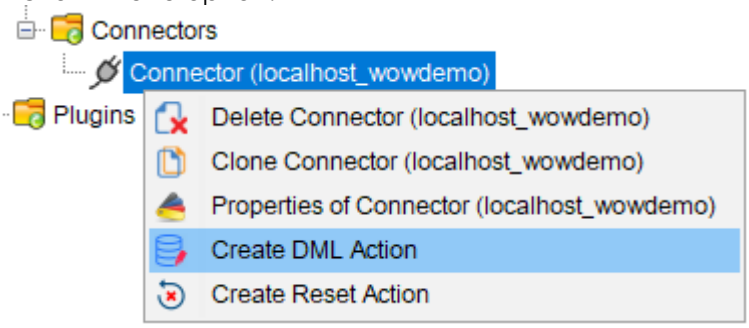


Figure 97 – DML Creation Menu

It creates the Action automatically. DML Actions execute RDBMS DML statements (INSERT, UPDATE, CREATE, TRUNCATE, …). Auto-Commit is ON. It takes only one parameter, i.e. the SQL statement as an expression.

3.9.6 Prepared DML Action

JDBC supports Prepared Statements as a faster alternative if same statement with different values are executed many times. To take advantage of the performance benefits, and in similar situations, a Prepared DML Action may be used instead of a normal DML Action.

Prepared DML Actions are created by right-clicking the Actions tree folder:

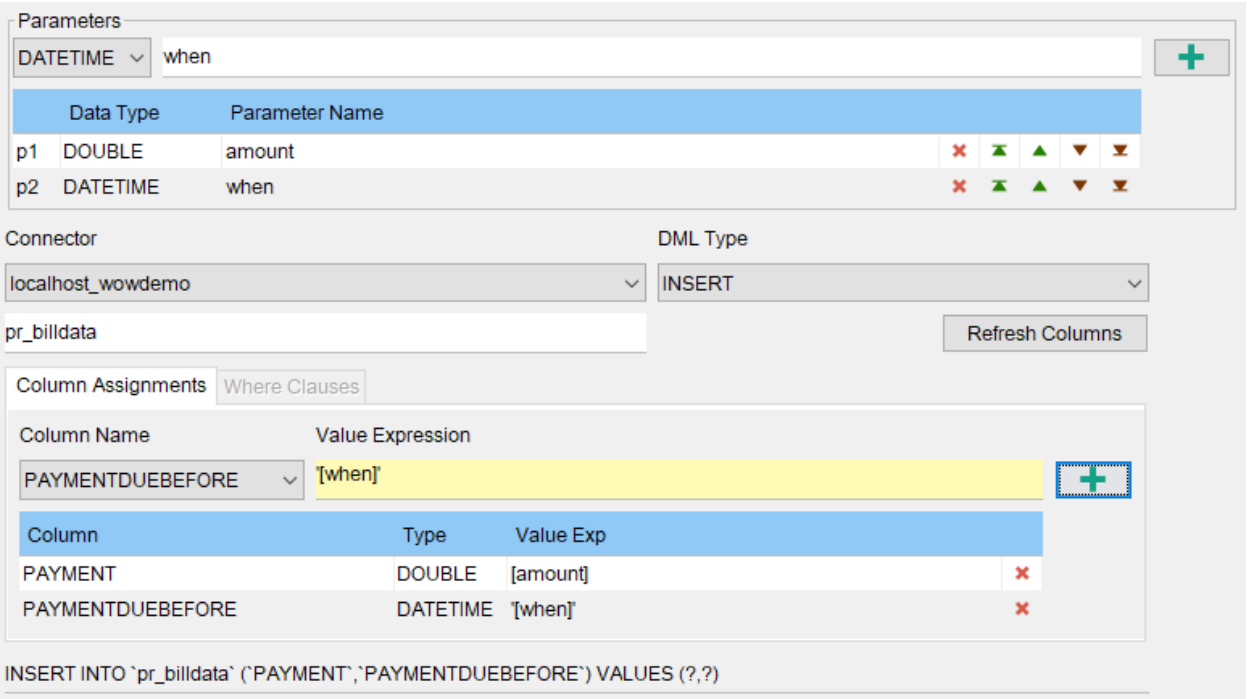


Figure 98 - Prepared DML Action editor

Prepared DML Action is a guided query generation environment, where the UI helps with selecting the column for value assignments and selecting the 'where' conditions. The difference in execution is that the actual values (captured in the expressions) as delivered separately at run-time to a Prepared Statement thus driving up speed.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Connector	A pre-configured DB Connector that will be used to execute the Prepared Statement.
DML Type	Only 2 types of DML queries are supported: INSERT and UPDATE. Note: for other types such as CREATE, DROP etc, use the normal DML Action where its possible to write the query as desired.
Table Name	The DB Table on which the INSERT or UPDATE query will execute. Note: Once the table name is changed, you may experience a temporary lag in the UI, as the Column Selection dropdowns are being re-populated in the background.
Column Assignments	Set the column wise value assignments here. For INSERT queries, it may be required to configure new values for all columns. For UPDARE queries, only a few columns may be updated.
Where Clauses	These are applicable only in case of UPDATE queries, where the 'WHERE' conditions can be defined.
Test	Executes the Action, and shows a completion confirmation message after execution of the Action is completed.

3.9.7 Stats Refresh Action

Figure 99 – Stats Refresh Action Editor

Stats Refresh Actions are created from the Right Click Menu of a Data Source. It triggers the Stats Data gathering thread for a Data Source. This Action must be called at least once before calling the Stats Functions for the corresponding Data Source Fields.

3.9.8 Socket Write Action

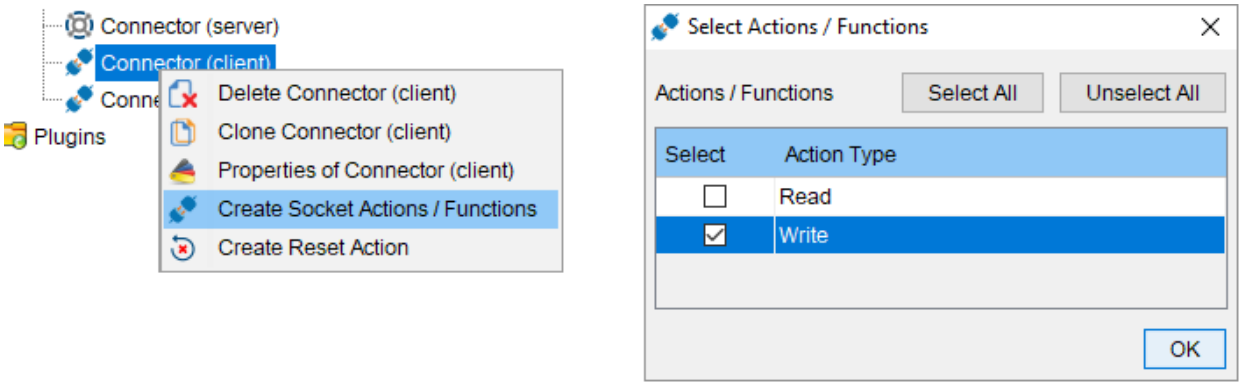


Figure 100 - Create Socket Write Action

The Socket Write Action writes a String into a connected client TCP Socket or a UDP Datagram Socket. This Action depends on the Client Socket Connector.

3.9.9 Server Socket Write Action

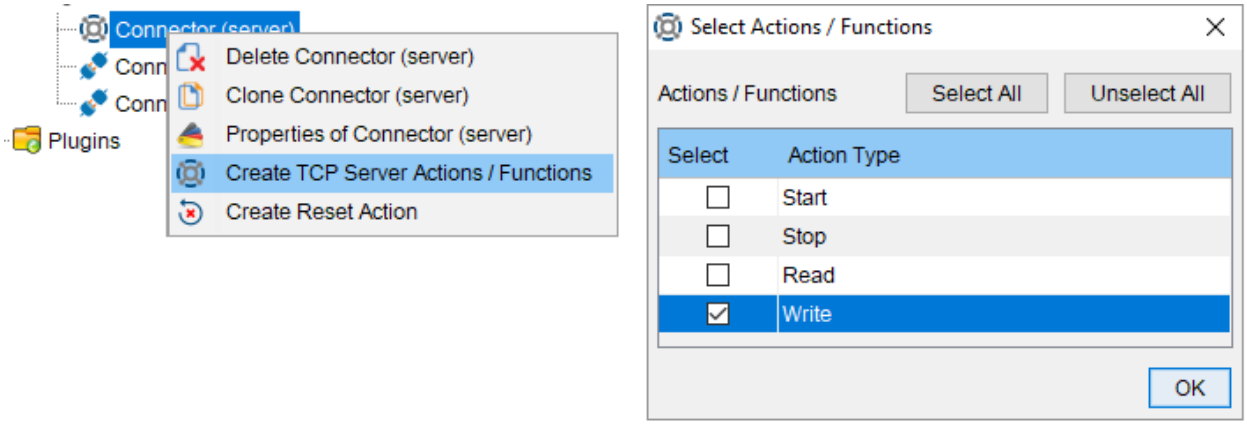


Figure 101 - Create Server Socket Write Action

The Server Socket writer outputs a String on a connected socket when a remote client connects to the built-in TCP server in PARE. It depends on the Server Socket Connector. Server Socket Write Action depends on the TCP Server Connector.

3.9.10 TCP Server Start Action

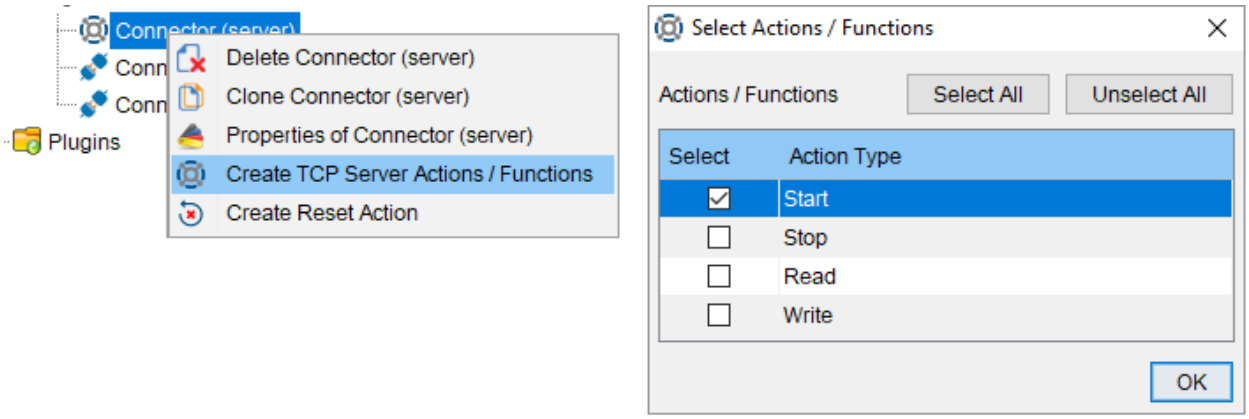


Figure 102 - Create TCP Server Start Action

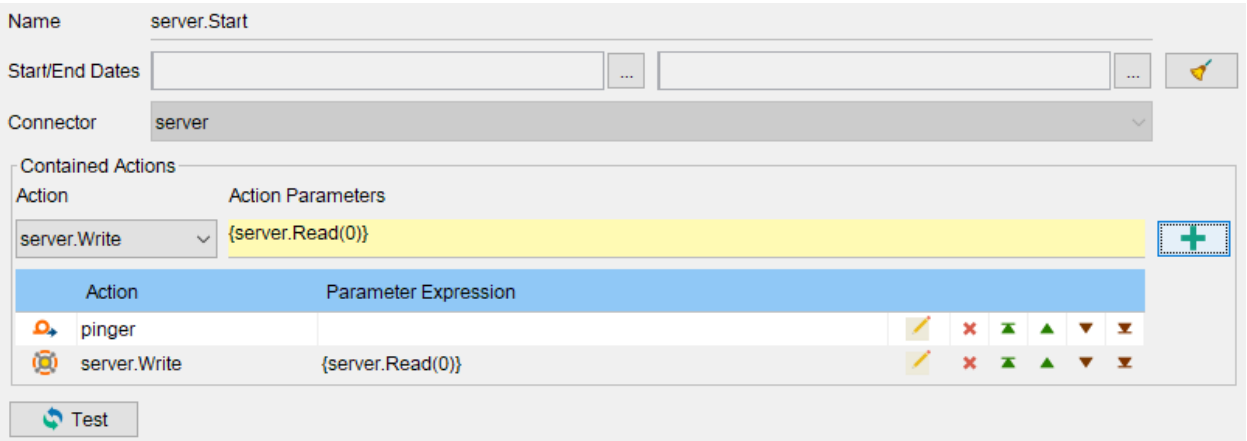


Figure 103 - TCP Server editor

The TCP Server Start Action launches a Server thread that listens for a connection request from remote clients. Once a connection is received, the contained Actions are launched in a separate thread per connection. This ensures that the connections are served in parallel and clients do not have to wait for other connections to finish. The I/O on the connection is achieved by including the Server Socket Read and Write actions in the contained Action hierarchy. In the above example, a simple 'ping' functionality is implemented by writing to the connected socket the String received from the same connection.

Only one server can be configured on a single Server Socket connector.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Contained Actions	Actions that are executed in sequence. The order of the actions can be changed by the adjoining buttons beside the Actions table view. The subscriber position buttons include: • Move up • Move down • Move to top • Move to bottom • Delete
Test	Executes the Action, and shows a completion confirmation message after execution of the Action is completed. **Note**: Clicking the test button starts the Server and returns immediately. The server keeps running in the

	background, until the 'Stop' action is invoked.

3.9.11 TCP Server Stop Action

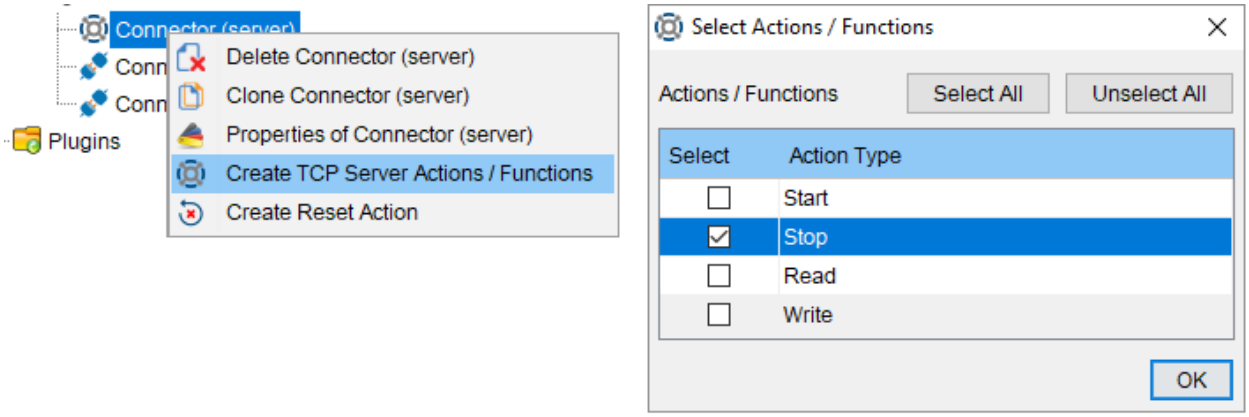


Figure 104 - Create TCP Server Stop Action

Stops an already started TCP Server. This Action brings down the background Server thread gracefully. However, the already active connections originated by this server continues to remain active and operate as usual. New connections to the server are not possible once Stop Action is invoked.

To bring down the server as well as close all the live Connections that this server has accepted, the Connector Reset Action needs to be invoked. It not only stops the server, but kills all the Connected Sockets as well.

3.9.12 Rule

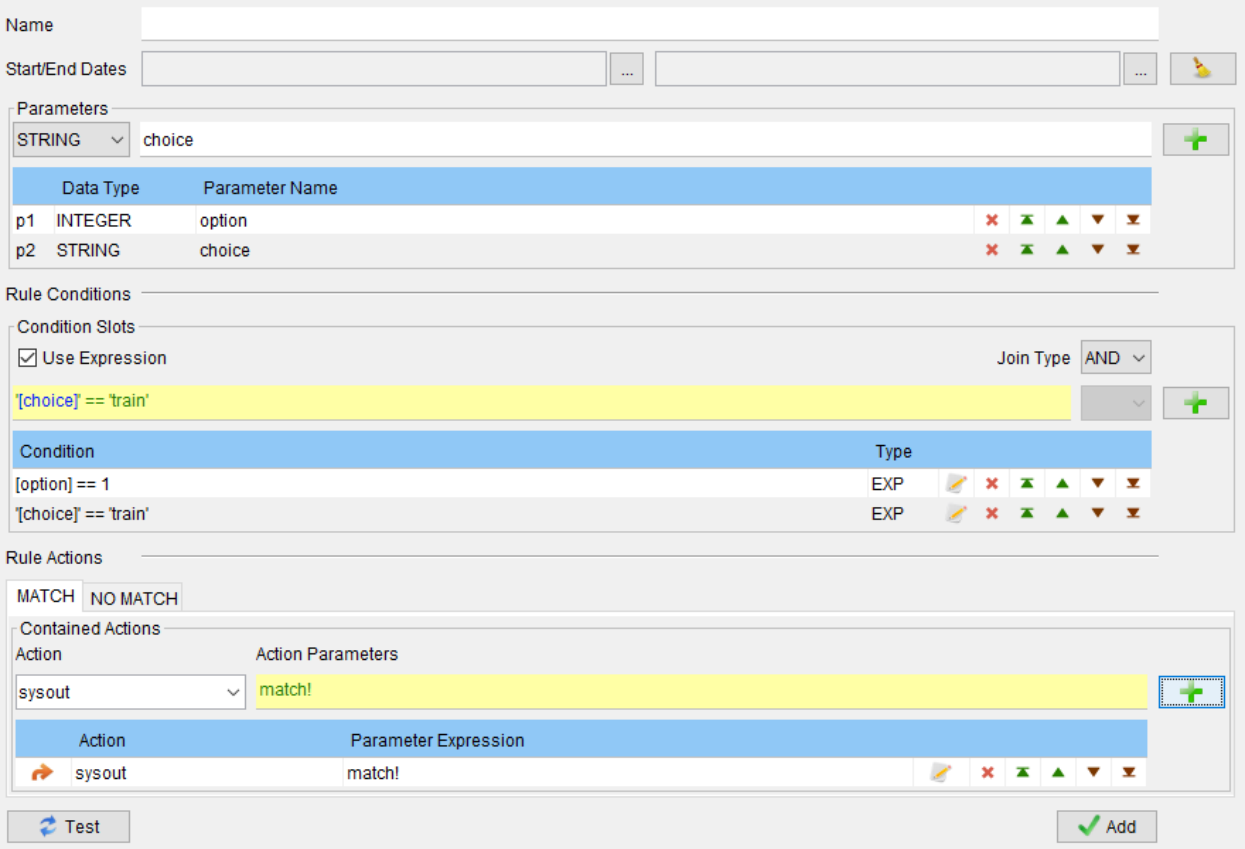


Figure 105 – Rule editor

A Rule Action divides the execution path of Actions based on outcome of set of pre-defined Conditions or condition expressions. One or more Rule Actions can be chained together to configure complex decision trees.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Condition Slots	One or more conditions can be joined (AND or OR) here. Conditions can be defined in 2 ways: Expressions – this is suitable for simple conditions and makes it more readable Pre-defined – pre-defined conditions can be used in this case and it is suitable for more complex conditions Note: Boolean functions can also be directly used (for example a script function with a Boolean return type)
MATCH Actions	Actions that will be called if the Condition Slots evaluate to TRUE.
NO MATCH Actions	Actions that will be called if the Condition Slots evaluate to FALSE.
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

A Rule Action can additionally inform the caller about outcome of the Condition Slots at execution time using the RuleListener callback (more on this in the API section). For example, users can run a sequence of Rules with a break-on-success condition attached. In such cases, this callback provides required information at run time.

3.9.13 Sequence

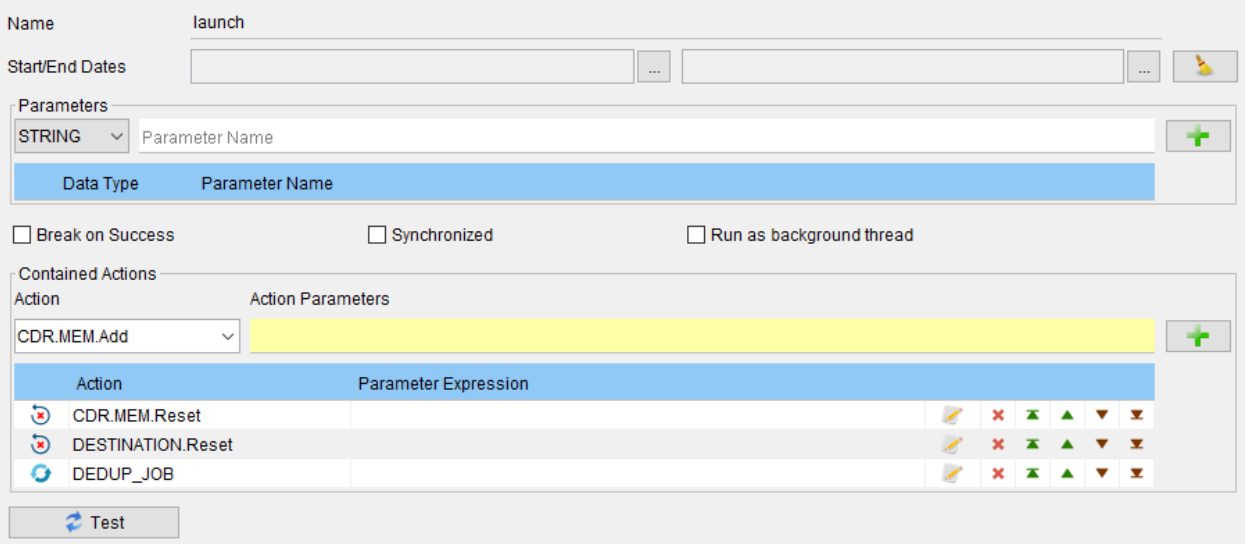


Figure 106 – Sequence editor

A Sequence is just a set of Actions in an ordered list. Actions can be moved up and down the list, and the order of execution will be maintained when the sequence is executed.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.

Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Break on Success	This flag signifies "rule only mode". If the Break on Success flag is checked, Sequence enters Rule Only mode. In this scenario, only Rule Actions can be added to the sequence. If you check Break on Success after you have already entered other Actions, at that point, all non-Rule Actions are removed from the Sequence. This flag can be set only if all the contained actions are Rules. If this is 'checked', execution stops (the Sequence returns) when a contained Rule's Condition Slots evaluate to TRUE (or MATCH).
Run as background thread	The contained Actions are executed in sequence under a background thread without blocking the calling thread. This facility should be used to launch long running background tasks – typically for housekeeping purposes. Note: This feature may be deprecated in future, and it's recommended that PARE Processes be used for such purposes.
Synchronized	This flag synchronizes execution of the sequence within the PARE Instance. If a synchronized sequence is called by more than one thread in parallel, they will be serialized. Cluster-wide synchronization: If Cluster-wide locking is configured and enabled (refer to <>), this option automatically enables cluster-wide synchronization. This helps in serialization in a multi-server setup and also cross engine synchronization.
Contained Actions	Actions that are executed in sequence. The order of the actions can be changed by the adjoining buttons beside the Actions table view. The subscriber position buttons include: • Move up • Move down • Move to top • Move to bottom • Delete
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

3.9.14 Transaction Action

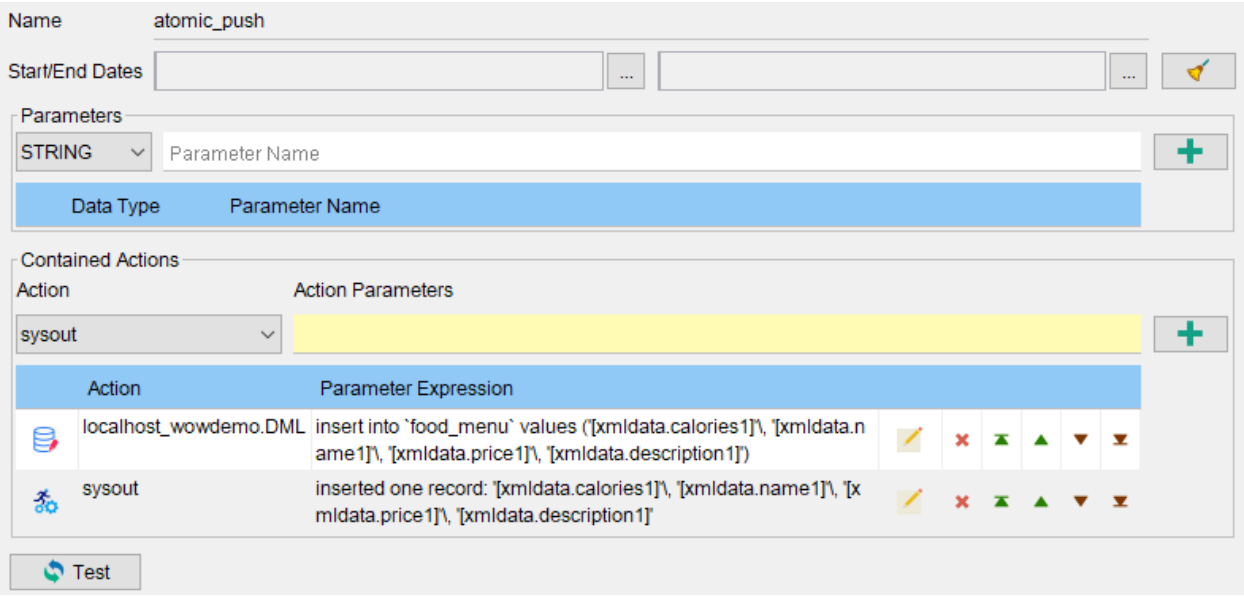


Figure 107 - Transaction Action editor

Transaction Action runs the contained actions under a Global XA Transaction. XA Aware actions like DML and PDML Actions automatically joins the Global Transaction and executes as a single atomic block. In case any of the child actions encountered an error, the entire Transaction is rolled back. All DML and PDML actions in the entire child hierarchy automatically falls under the same global transaction.

Note:

- In case a Transaction Action is embedded within another Transaction Action, the parent Transaction is used – no separate nested transaction is created.
- A Transaction does not propagate to multiple threads. Hence automatic enlisting of JDBC connections happen when the Connections are used in same thread where the Transaction Action is running.
- Multiple parallel threads can execute multiple separate global transactions. Hence, it's possible to create a MT Job, where the contained actions can be Transaction Actions.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Contained Actions	Actions that are executed in sequence. The order of the actions can be changed by the adjoining buttons beside the Actions table view. The subscriber position buttons include: • Move up • Move down • Move to top • Move to bottom • Delete
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

3.9.15 Job

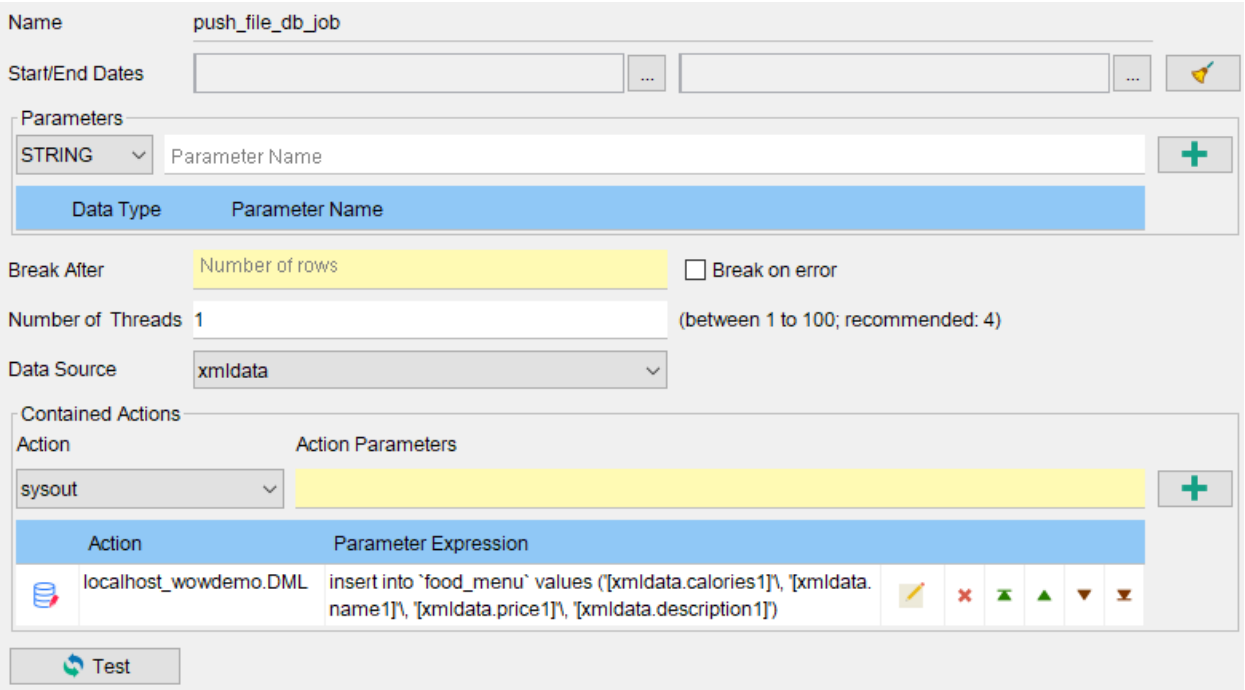


Figure 108 – Job editor

A Job iterates a data source. For example, it can automatically iterate through all lines of a Data Source and execute a set of Actions. This is useful as normally Rule Processing is done at record level and these records belong to some data source. As an example, a CDR file may be mapped as Data Source and required Fields may have been mapped from it. Now processing the CDR file to calculate call charges or updating call stats may be done by configuring a Job.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Break on Error	If this option is checked: • In Single threaded mode: The job returns pre-maturely before finishing processing of remaining Data Source rows. • In Multi-threaded mode: The thread that faced the error returns immediately. However other threads continue to perform as usual. It is therefore recommended that for MT Jobs, un-check Break on Error, and use On-Error Actions instead.
Break After	Maximum number of rows to be processed by the Job. If it's set to ZERO or Blank, ALL rows of the Data Source are processed. This is commonly used for testing purposes on large data sources.
Number of Threads	Default value is ONE and that indicates single threaded operation. If a value > 1 is specified, the Job executed in MT mode.
Data source	The Data Source that the Job will loop (either in single or MT mode).
Actions	The child Actions that the Job will fire per row of the Data Source.
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

3.9.15.1 Multi-Threaded Execution

Jobs support multi-threaded execution of configured actions. It creates a thread pool as per configured number of threads and distributes the actions to the threads in parallel. It helps in much faster execution and better CPU/Memory utilization when processing large data sets.

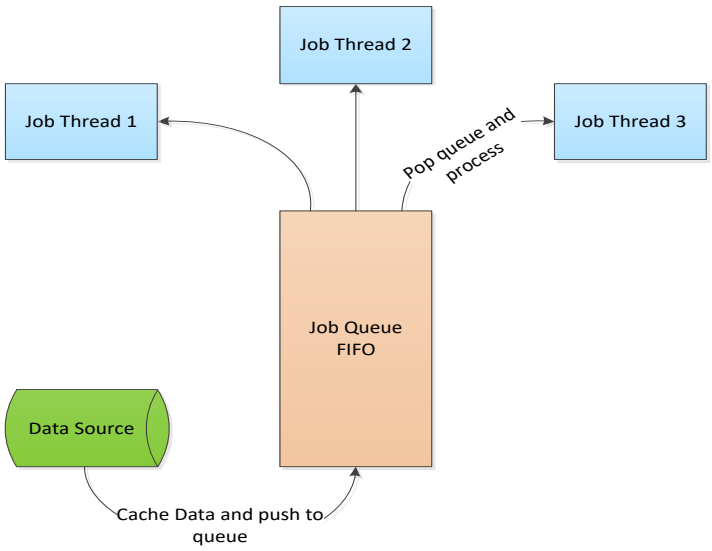


Figure 109 - Multi threaded Job processing

Please note:

- Multi-threaded execution may increase memory consumption and so appropriate testing in your environment is recommended.
- Even though the contained Actions are executed in parallel, the Job returns only after all the threads have finished executing.
- If a DB Data Source is being used, it's recommended that the Connection Pool be sized so that each of the threads can get to work with a separate Connection so that execution can happen in parallel.
- As with any multi-threaded application, the gain in performance is not linear. Since thread scheduling itself has additional overheads, and since the OS ultimately assigns free Cores to threads, it's natural to expect diminished performance gains as we keep increasing the number of threads. A representative graph based on actual performance of a sample workload is shown below as example:

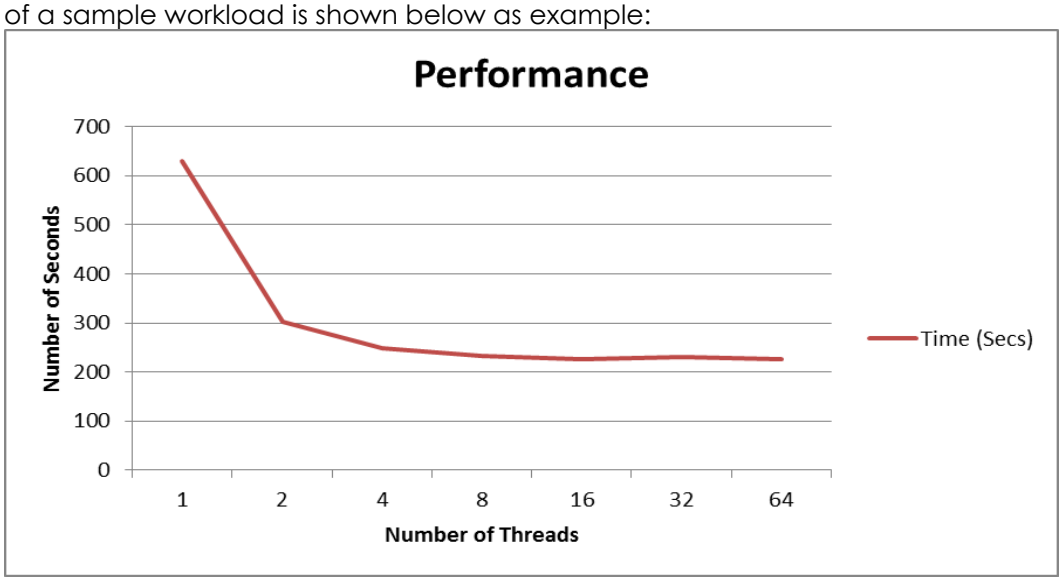


Figure 110 – Number of threads / times

- If a Custom Action (backed by a Class Connector) is executed in MT mode, the Object should make sure that internal data used by the object are handled in a thread-safe manner.

Note that in MT mode:

- Break on error flag does not work
- Built-in Break action does not work

3.9.16 Loop Action

Loop Action implements a loop container with a fixed number of iterations. It's like FOR or DO loop functionality found in many programming languages. It can be created as follows:

Loop actions can accept parameters, and also provides a symbol for the current loop index ([<action name>.index]). This Action can be used to iterate a fixed number of rows from a Data Source, or to call the set of contained Actions a specific number of times. Note that the min, max indexes as well as the increment intervals are regular PARE expressions.

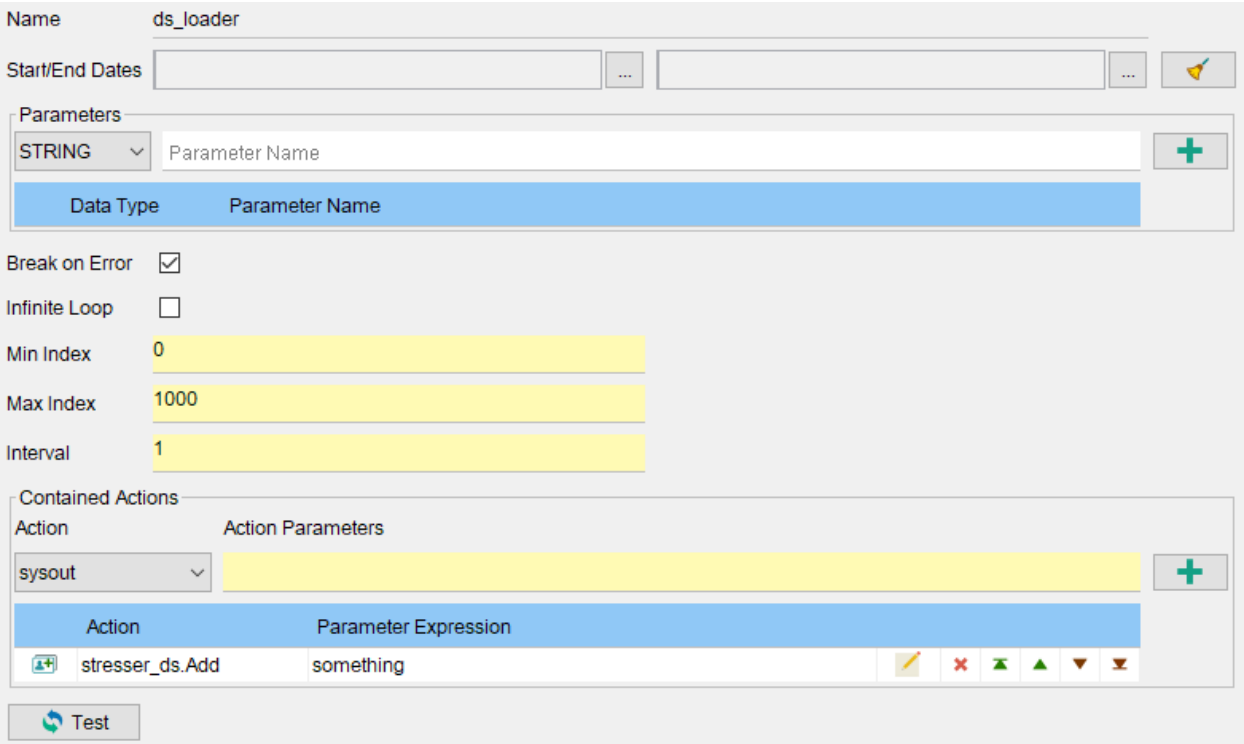


Figure 111 - Loop Action editor

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Break on Error	If this option is checked the loop returns pre-maturely before finishing processing of remaining iterations.
Infinite Loop	Starts an infinite loop: each time executing all the contained Actions. An index is available – starts at ZERO and goes up to 2147483647 before being reset to 0 again.
Min index	Loop iteration to start from this index (expression).
Max index	Loop iteration to continue until this index (expression). Note: - Iteration may be aborted in 'Break on error' is checked and an error is encountered - Max index must be a greater number that Min index
Interval	Index increments per iteration. For example, if the Min index is 1, Max index is 10 and Interval is 2, the Loop Action will fire the Contained Actions 5 times for indexes 1, 3, 5, 7 and 9.
Actions	The child Actions that the Job will fire per row of the Data Source.
Test	Executes the Action, and shows a completion confirmation message after execution of the Action is completed.

3.9.17 Field Setter Action

Setter Actions set value of a single field. Setter Actions can be created only via the corresponding Field's Right click menu and name is set automatically. Please refer to Figure 8 for the menu option. It takes only one parameter and that is the value of the field in String format. Action containers like Rules, Jobs or Sequences can include this Action to set individual field values.

The difference with Field Action is that here the field value is passed as a parameter to the Action rather than hard-wired in the Field Action definition.

Note: Only User, App and Data Source type of Fields can be set using this Action.

3.9.18 Decision Table

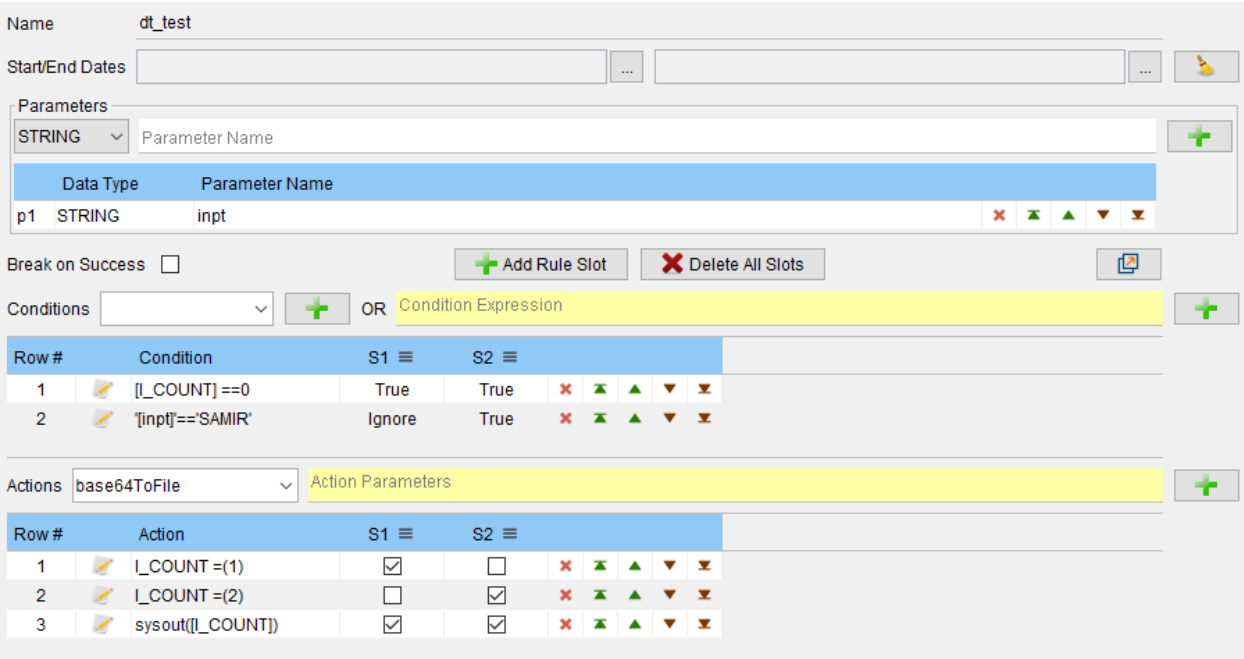


Figure 112 – Decision Table editor

The concept of Decision Table can be looked up here. PARE enables you to connect multiple Conditions and Actions via a Decision table. Unlimited number of Rule Slots, Conditions and Actions can be configured, making it a powerful tool for Business Users to define application behavior.

With PARE Decision Tables, multiple Rule Slots can be created. Each slot corresponds to one or more Conditions, which if TRUE, triggers one or more Actions.

A condition can be checked into a slot by marking it as TRUE or FALSE. If IGNORE is set for a particular condition, it will not be evaluated when processing the slot. Note that if there is more than one Condition checked in one slot, it will be considered TRUE if all of the checked Conditions evaluate to the corresponding TRUE or FALSE outcome.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Break on Success	The DT will return upon first successful slot (all conditions of the slot are true and all corresponding Slot Actions are executed)
Add Condition	Add a Pre-defined condition to the DT.
Add Expression	Add a Condition expression to the DT. Note that, both Pre-defined conditions and Condition Expressions can be added as separate rows in the DT. For very simple checks (and where Mandatory flag is not applicable), using Condition Expressions results in better readability. In

	other cases, using pre-defined conditions are recommended.
Add Action	Adds Actions to the DT. Each Action is a separate row in the Action table. Actions can be added with parameters as PARE expressions.
Add Rule Slot	Adds a Slot in both the Conditions and Actions tables. Required number of check-boxes for selecting the desired combination is also automatically created.
Delete all slots	DT can be complex, and if you feel it's all messed up, use the 'Delete All Slots' button to nuke the configuration and start from scratch.
Configuring the Slots	Basically, user needs to check the right combination of Conditions (True / False / Ignore) and Actions per Slot. Each of the conditions is checked in their separate slot. It's common to have more than one Condition checked (set as True or False) in a single slot. Similarly, Actions should be checked per Slot. **Note**: When the DT is executed, each slot is executed one by one, and in sequence.
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

3.9.19 Case Table Action

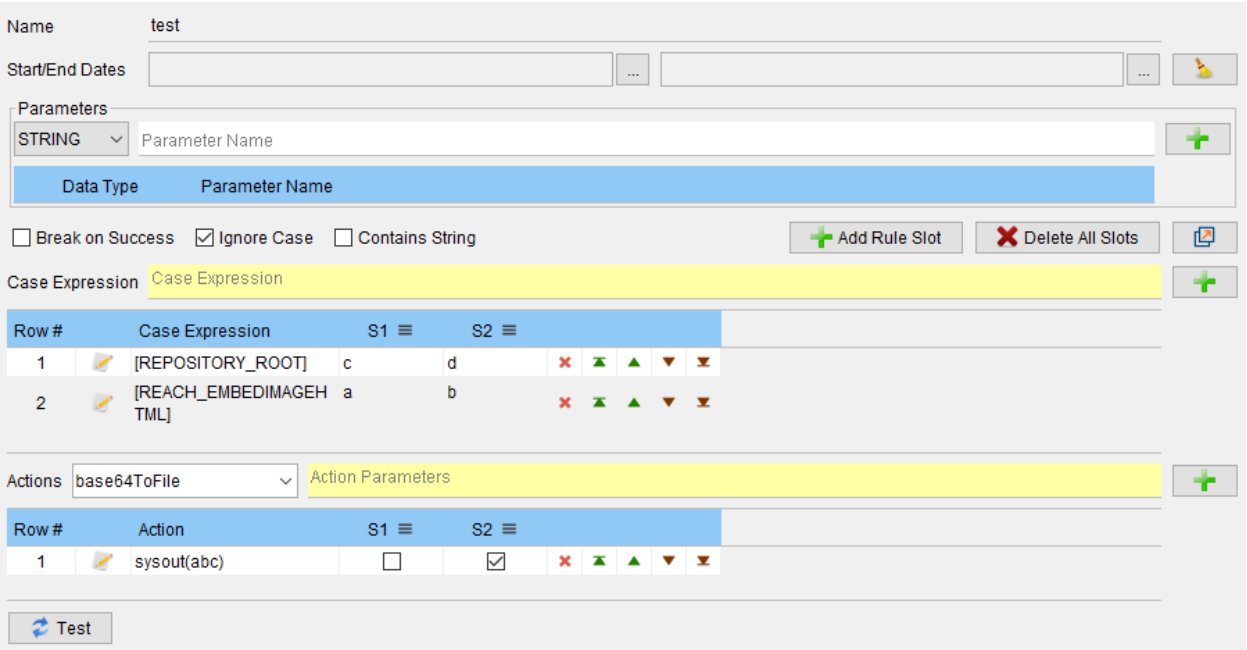


Figure 113 - Case Table Editor

This is a variant of the Decision Table where instead of random condition slots, there are value comparison slots – much like a "case table" (CT) – where a combination of one or more values form a business case for a particular set of actions. Case Tables configurations are more compact and easier to read than Decision Tables and therefore should be used in place wherever possible.

The remaining characteristics and behavior are like Decision Tables and hence may be referred to in the previous section.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter

	Description.
Break on Success	The CT will return upon first successful slot (all case values match for the slot and all corresponding Slot Actions are executed)
Ignore Case	Ignore case when comparing values with the case expression values
Contains String	If checked, the comparison method checks if the Slot value exists within the case expression. Otherwise, the slot value should be equal to the case expression value.
Add Expression	Add a Value expression to the CT. Note that for each Rule Slot, the actual value of the expression will have to be entered in the Case Table. This can be done directly by entering value in the table cell.
Add Action	Adds Actions to the CT. Each Action is a separate row in the Action table. Actions can be added with parameters as PARE expressions.
Add Rule Slot	Adds a Slot in both the Cases and Actions tables.
Delete all slots	CT can be complex, and if you feel it's all messed up, use the 'Delete All Slots' button to nuke the configuration and start from scratch.
Configuring the Slots	Basically, user needs to check the right combination of Cases (value comparison) and Actions per Slot. Each of the values are checked in their separate slot. It's common to have more than one Value configured in a single slot. Similarly, multiple Actions can be checked per Slot. **Note**: When the CT is executed, each slot is executed one by one, and in sequence.
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

3.9.20 On Error Action

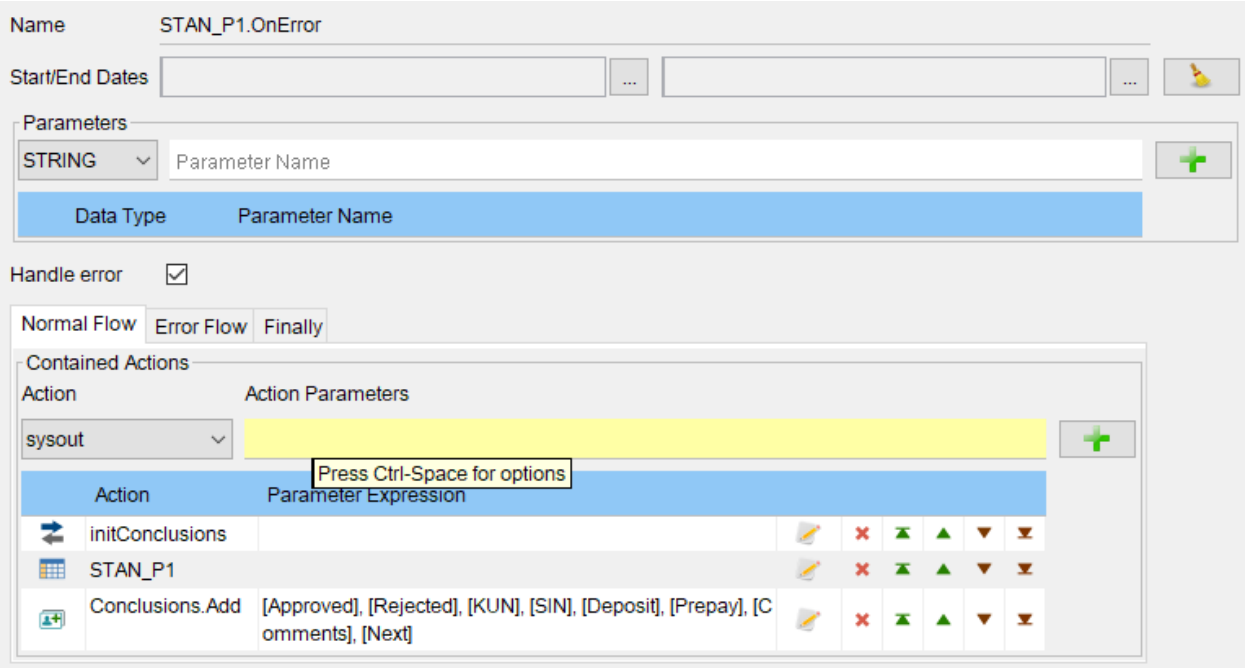


Figure 114 – On Error Action editor

On Error Action is an action container which wraps one of more other Actions within an error handler. If any error occurs when executing one of the contained Actions, a 2nd list of Actions is fired. The 'finally' container executes in either case.

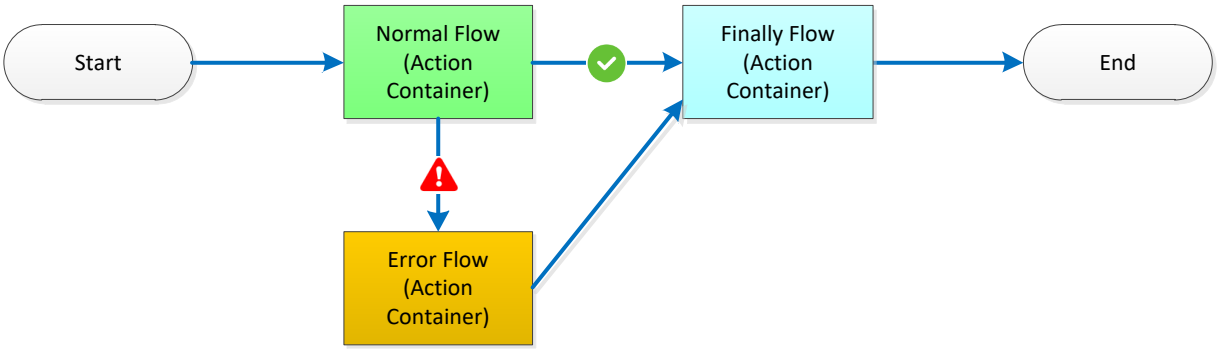


Figure 115 - On-Error Action flow

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Handle Error	If unchecked, error handling is bypassed – in this case the Action behaves more like a Sequence.
NORMAL Actions	Add Actions (and their parameters) to the OnError wrapper. These will be executed, and any error encountered will trigger processing of the ON ERROR Actions.
ON ERROR Actions	Add Actions (and their parameters) that will be fired if any of the Normal Actions throw an Exception.
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

Note: It should be noted that On Error Action can recover from a recoverable error. Fatal run-time exceptions like low–memory errors or low-level hardware errors like low disk space etc., cannot be effectively handled by this facility.

3.9.21 Scheduled Action

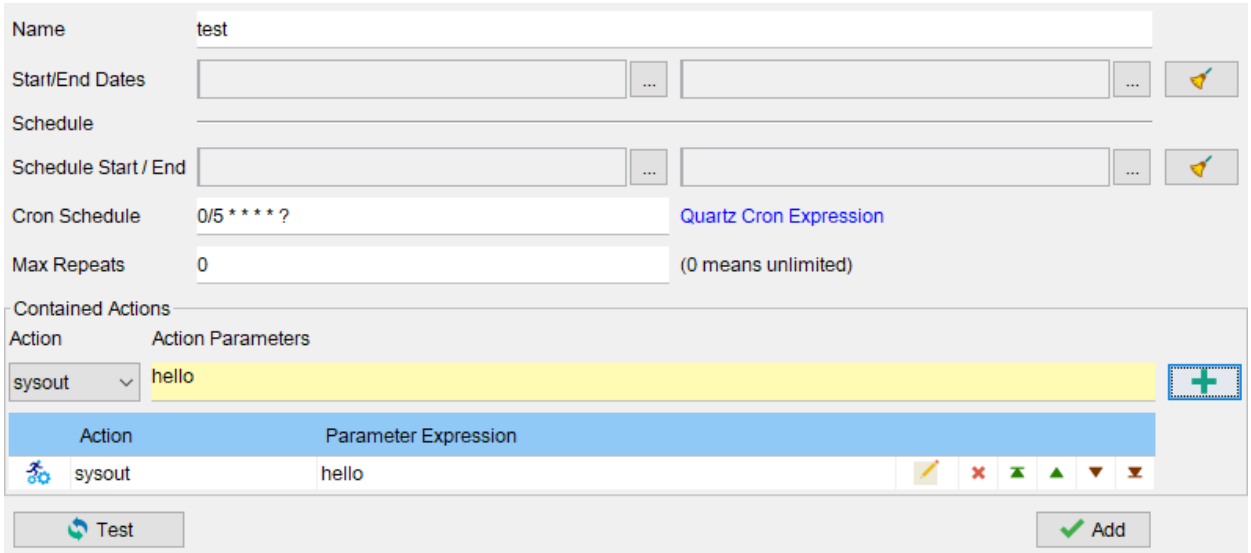


Figure 116 - Scheduled Action editor

Scheduled Action allows automatic background execution of an Action as per a schedule defined by the 'CRON' expression. Additionally, users can set start and end dates (optional) and maximum repeat counts (how many times the Action will be executed).

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Schedule Start / End	Date window within which the Scheduler will execute. By default these dates are not set, and the default meaning is: • No Start date: Start scheduling immediately • No End date: Carry on scheduling till infinity
CRON Expression	CRON expression defining the interval of scheduling. Please refer to CRON documentation (Cron Wiki) for full description of the options that can set using this expression.
Max Repeats	A repeat count can be set to restrict the total number of executions.
Child Actions	Actions that are executed in sequence. The order of the actions can be changed by the adjoining buttons beside the Actions table view. The subscriber position buttons include: • Move up • Move down • Move to top • Move to bottom • Delete
Test	Executes the Action, and shows a completion confirmation message after execution of the Action is completed.

Note: Quartz Scheduler (http://www.quartz-scheduler.org/) is used by PARE as the Action scheduler.

3.9.22 Script Actions

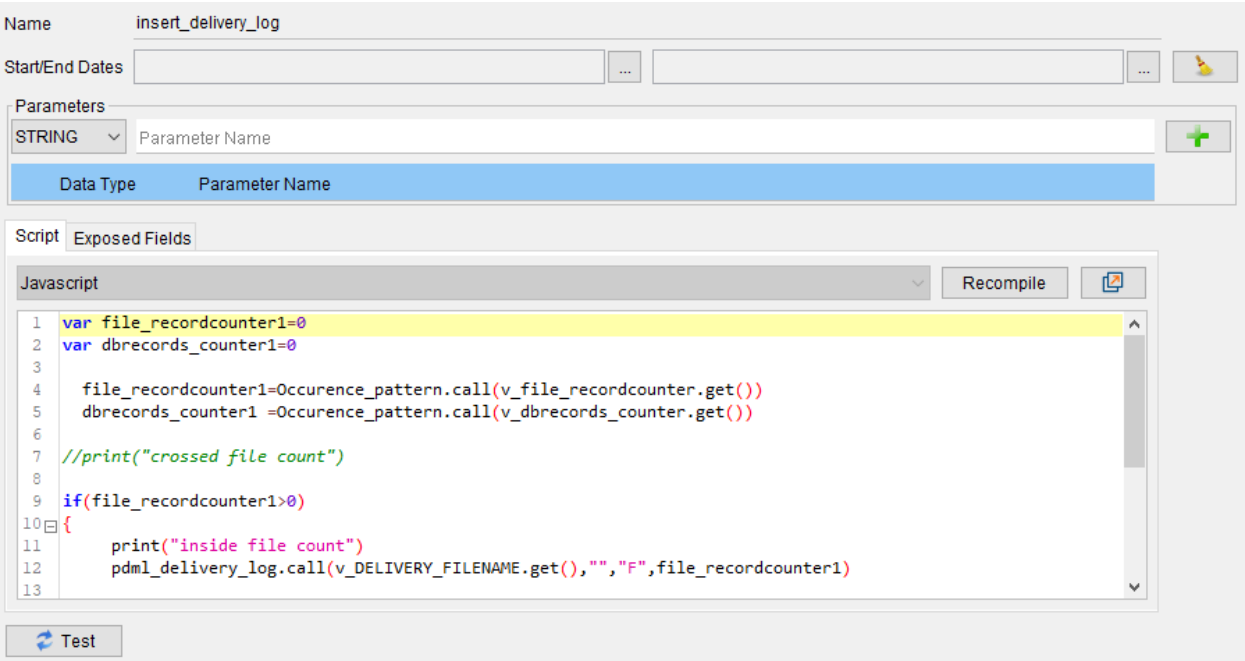


Figure 117 - Script Action editor

Script Actions enables execution of Scripts as Actions. This feature greatly enhances ability to implement complex algorithms inside PARE instead of writing custom Actions or Functions in Java. Please refer to section 3.4 for details about the Script architecture.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Select Field	Select Fields one by one to expose in the Script
Select All	Select ALL PARE fields to expose in the Script
Exposed Fields	PARE Fields exposed (copied) within the Script
Script Type	JavaScript or LUA (other languages may be added in future)
Script Editor	Syntax aware editor for writing the Scripts.
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

3.9.23 Email Send Action

Email Send Action is designed to send an Email using a pre-configured SMTP Connector. It can embed custom headers within the email – for later tracking, configurable attachments and either plain text or HTML email body. This Action with all the options configured, can act as an email template that can be re-used with the process flow. One or more such Actions can be created per SMTP connector.

Email Send Action can be created using the right click option menu.

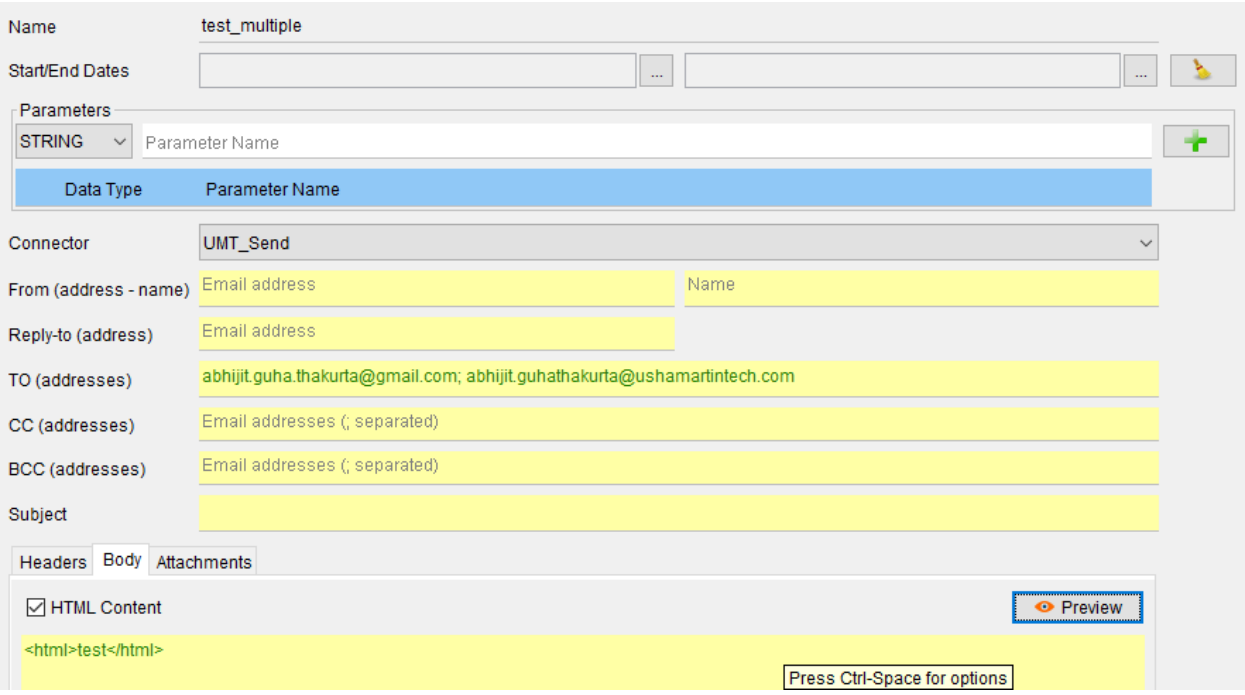


Figure 118 – Email Send Action editor

The Action supports PARE expressions to be set for From, To, Subject, Headers, Body and Attachment filenames.

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Connector	The pre-configured SMTP connector that is used to deliver emails by this Action
From (address)	Email address that will appear in the FROM section of the email message. This will also serve as the default reply-to address.
From Name	Name of the Sender that will appear in the email
Reply-to (address)	The default reply-to address can be overridden using this field
To (addresses)	Recipient email addresses. Multiple addresses can be entered with a ; in between
CC (addresses)	CC email addresses. Multiple addresses can be entered with a ; in between
BCC (addresses)	BCC email addresses. Multiple addresses can be entered with a ; in between
Subject	Email subject text expression
Headers	Email headers (name/value pairs) that will be attached with the message. This is useful as the same be traced in replied and/or bounced emails to establish traceability.
Body	Body text expression. Can be interpreted optionally as plain-text or HTML. Since PARE expressions are supported, users can easily embed external files using the {Builtin.filetotext()} function.
Attachments	List of files that will be attached with the email. All attachments are embedded as octet stream in a multi-part email message.
Test	Executes the Action and shows a completion confirmation message after execution of the Action is completed.

3.9.24 Parallel Tasks Action

A Parallel Tasks Action is very similar to a Sequence, with the key difference being:

- All the Contained Actions are launched as separate independent threads

This Container can be used to launch parallel tasks and wait till all of them are completed. Note that the parameters passed to such actions are resolved in the main thread.

3.9.25 Event Action

An Event Action signifies a business event (ex: a payment is received, or a new PO is generated), that can be embedded within PARE Orchestrated flows. A single flow may optionally contain one or more Event Actions. Event Actions are used to trigger webhooks during flow execution.

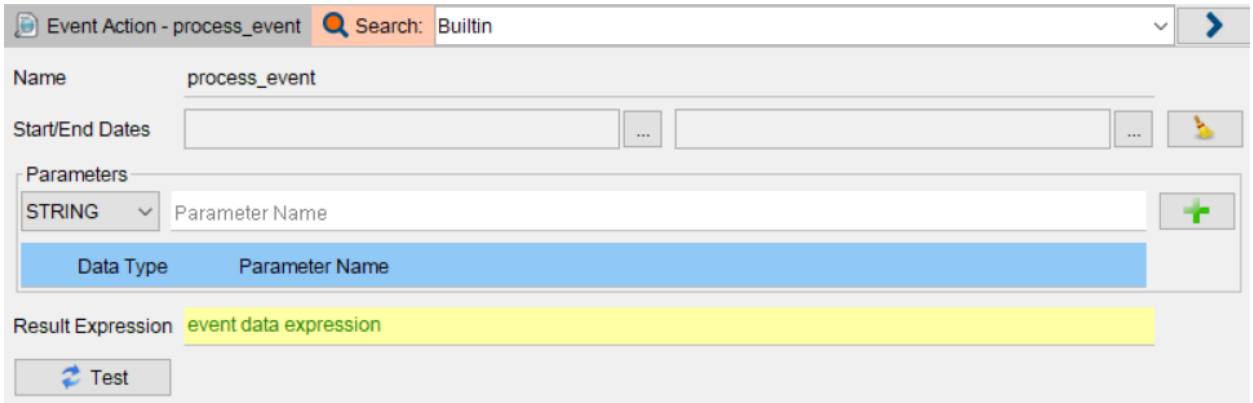


Figure 119 - Event Action editor

Name	Name of the Action
Start/End Dates	Date Window within which this action will fire. If both are set, and the Date of execution lies outside the window, the execution of the Action will be ignored.
Parameters	Definition of the different parameters that is required to call this Function. Each parameter is described by a Parameter Data Type and Parameter Description.
Result Expression	The Expression that will be evaluated at the time of firing the Event and sent by POST to all the registered webhook URLs.

The payload sent to the webhook URL has the following information (in JSON format):

- engine: name of the Engine
- event: name of the Event fired
- evtid: a unique event identifier
- result: value of the Result Expression configured in the Event Action
- evtime: timestamp when the event was fired
- username: username of caller (optional: in case a PARE Server API call triggered the event and an Access Token was passed to make the API call)

Note:

a) PARE will try to call every registered webhook once an event is fired. However, there is no guarantee that the webhooks payloads for sequential events will be delivered in the same sequence. Recipient systems should be prepared to receive out of order events.

b) PARE will make best effort to call every webhook. However, in case the call back URL is not active (server is down or network outage at the receiver end), it's possible to miss an event notification. If such cases, the recipient may search for the outcome (result) of a call using the events API endpoints described later in this document.

3.9.25.1 Monitor Events

PARE Server comes with a built-in "monitor" engine with certain pre-defined event actions. These can be used to get updated about following internal events:

Engine	Event Name	Operations	Payload
monitor	auth	login	source_ip username status (or / error)
monitor	workflows	startWorkflow taskStarted taskFinished taskCancelled taskAssigned instanceRemoved	Wfi Transfer Object

Engine	Event Name	Operations	Payload
monitor	processes	launched and killed operations on resources: process name	Launch command (for launched operation)
monitor	engines	POST / DELETE on following resources: engines	Name of the engine
monitor	users	GET / POST / PUT / DELETE on following resources: user local_user local_user_password local_me_password ldap_user sso_user user_attrs user_calendar user_leaves user_workgroup user_parent user_partition user_email user_mobile	Varies depending on operation.
monitor	workgroups	GET / POST / PUT / DELETE on following resources: workgroups workgroup_permissions workgroup_denylist	Varies depending on operation.
monitor	params	GET / POST / PUT / DELETE on following resources: params passwords settings	Name of the Host Parameter, Named Password or Setting.
monitor	ops	GET / POST / PUT / DELETE on following resources: user_block user_unblock vault_algo defaultfilestore tokenduration loglevel license mondb partitions cluster_enable cluster_disable authsources authidp holidays	Varies depending on operation.

3.9.26 Action Graphs

This facility lets the user view a flow diagram helping understand the entire process hierarchy. It's available for all Actions and can be accessed from the right-click menu:

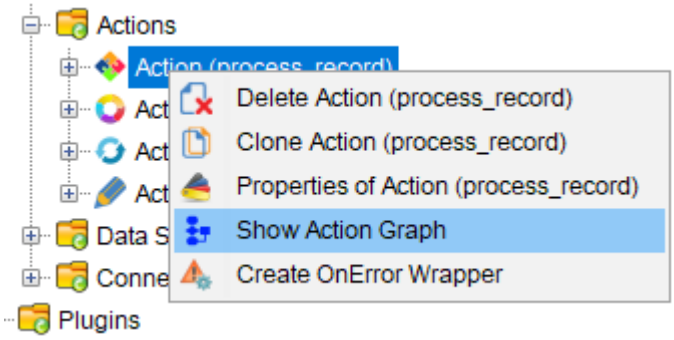


Figure 120 - Access the Process Graph

The Action graph viewer is a popup dialog, and each node in the diagram has an appropriate tool-tip to help understand the underlying data flow. Users can click on any of the displayed node to directly reach the editor screen or the corresponding node.

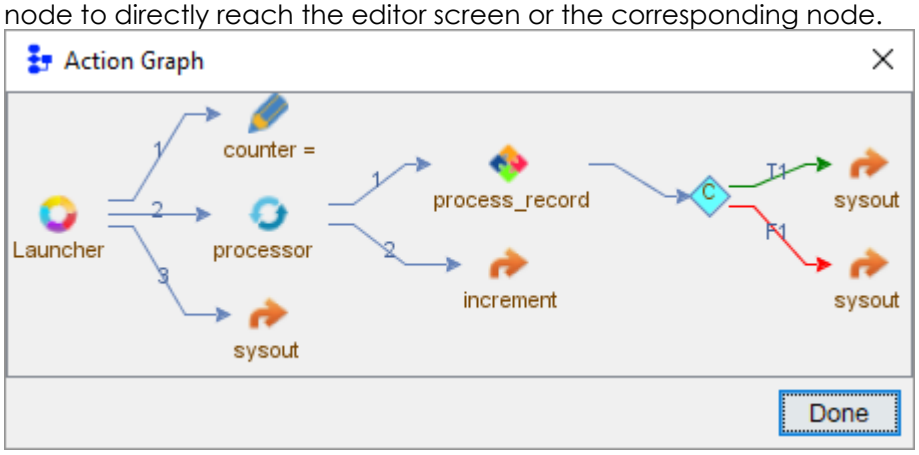


Figure 121 - Action Graph viewer

3.10 Connectors

Connectors implement interfaces to standard resources. PARE includes:

- DB Connector – a pooled JDBC connector to any supported database
- SMTP Connector – a connector to an SMTP server for sending emails
- POP3/IMAP Connector – a connector to email server for reading emails
- FTP Connector – a connector to a S/FTP server for upload / download of files
- Class Connector – a connector to custom Java resources (class files / Jar)
- Socket Connectors – for TCP and UDP communications

3.10.1 DB Connector

A DB Connector is used to configure the pooled connector to an RDBMS. Create the node as usual:

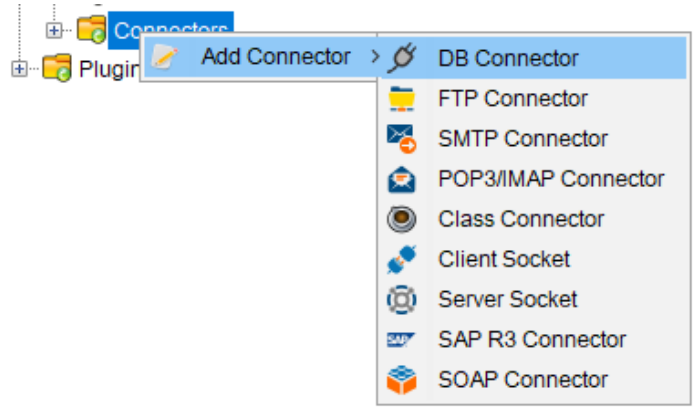


Figure 122 – Add Connector menu

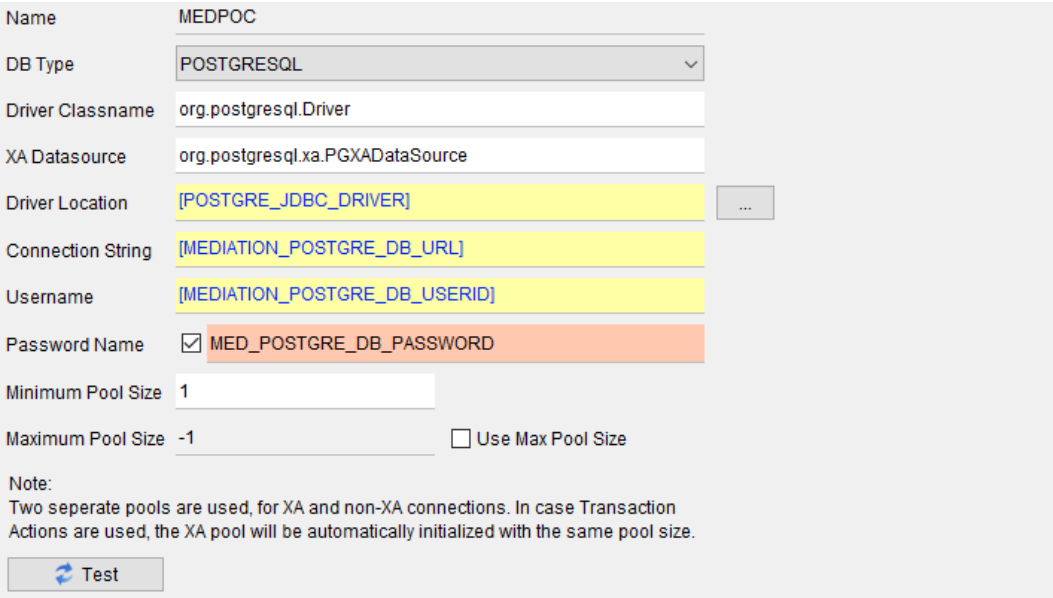


Figure 123 – DB Connector editor

A DB Connector node encapsulates all information to open an on-demand pooled connection to a RDBMS. PARE uses JDBC driver and Apache DBCP2 for DB access. The Minimum Pool size is the number of connections that the pool manager starts with. The Maximum Pool size is the number of connections after which, clients will be put on wait until an already allocated connection be returned to the pool.

Name	Name of the DB Connector
DB Type	All Databases which support JDBC are supported
Driver Classname	Driver class name to be loaded from the driver. Usually, the default values need not be changed. But sometimes, when using non-standard drivers, this value may need to be specified.
XA Datasource	XA Datasource class name to be loaded from the driver. Usually, the default values need not be changed. But sometimes, when using non-standard drivers, this value may need to be specified.
Driver Location	Location of the JAR file that implements the corresponding JDBC driver. Please refer JDBC documentation of your Database.
Connection String	JDBC connection string with host/port/schema name. Please refer JDBC documentation of your Database.
Username	Database access username. Some databases support putting the username and password in the Connection String itself. We do not recommend using that option for security purpose.
Password Name	Database access password name (refer to Password Manager documentation). Some databases support putting the username and password in the Connection String itself. We do not recommend using that option for security purpose.
Minimum Pool Size	Minimum number of connections created on first access. Managed by Apache DBCP 2.
Maximum Pool Size	Maximum number of connections created before blocking. Managed by Apache DBCP 2. If -1 is set (the default), there no upper limit to the number of connections than can be acquired. This is recommended setting for configurations deployed via Tomcat or other servlet containers.
Test	Connects to the database or shows an error.

3.10.2 DB Connector Link

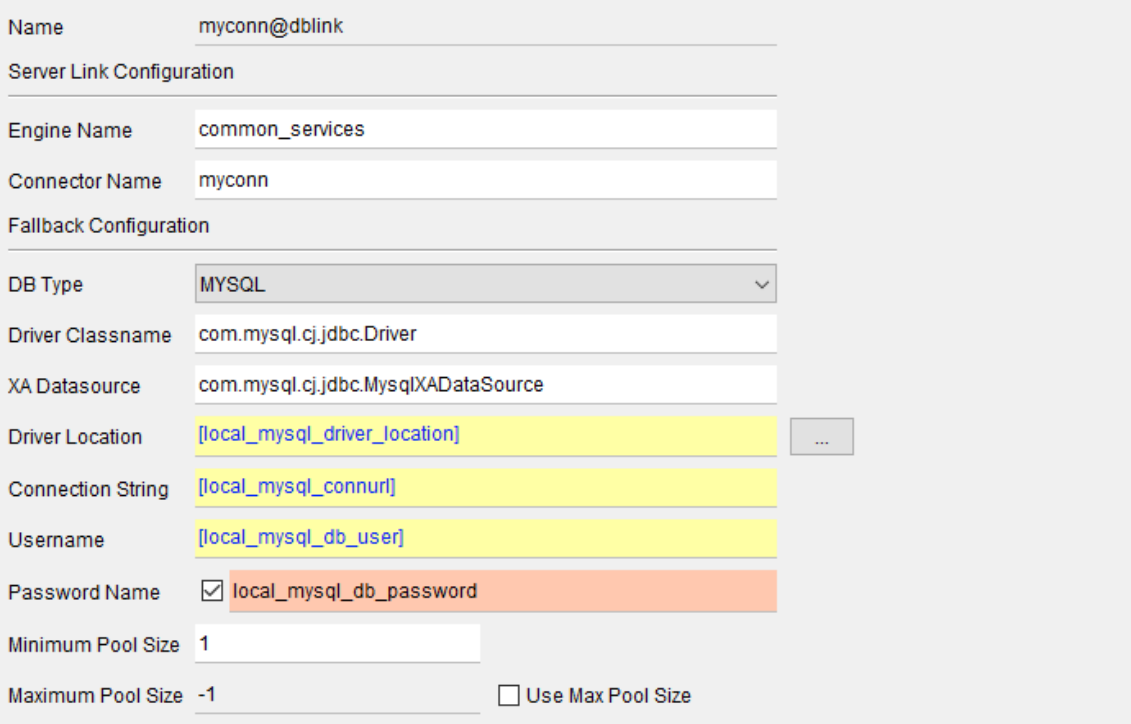


Figure 124 - DB Connector Link editor

Often multiple engines within the same PARE Server instance uses the same DB. Since a normal DB connector allocates DB connections per active thread per engine, this may cause many connections to remain open even though it's not being used (the allocated thread may be busy doing something else).

The solution to this problem is a system to share the same DB Connector across different engines within the same PARE Server. The DB Connector Link configuration offers this facility. This only works in PARE Server context and therefore it includes a "fallback" configuration, which imitates a normal DB Connector, when used outside of Server – for example, when used from within the Studio for testing purposes.

The editor has two (2) additional fields described below. Rest are same as a normal DB Connector and description of the same can be found in the previous section.

Engine Name	Name of the provider engine that has the normal DB Connector configuration
Connector Name	Name of the DB Connector within the provider engine

3.10.3 SMTP Connector

An SMTP connector manages a connection to a SMTP server (for outbound emails). It supports following encryption specs:

- SSL on connect
- TLS Enabled
- TLS Required

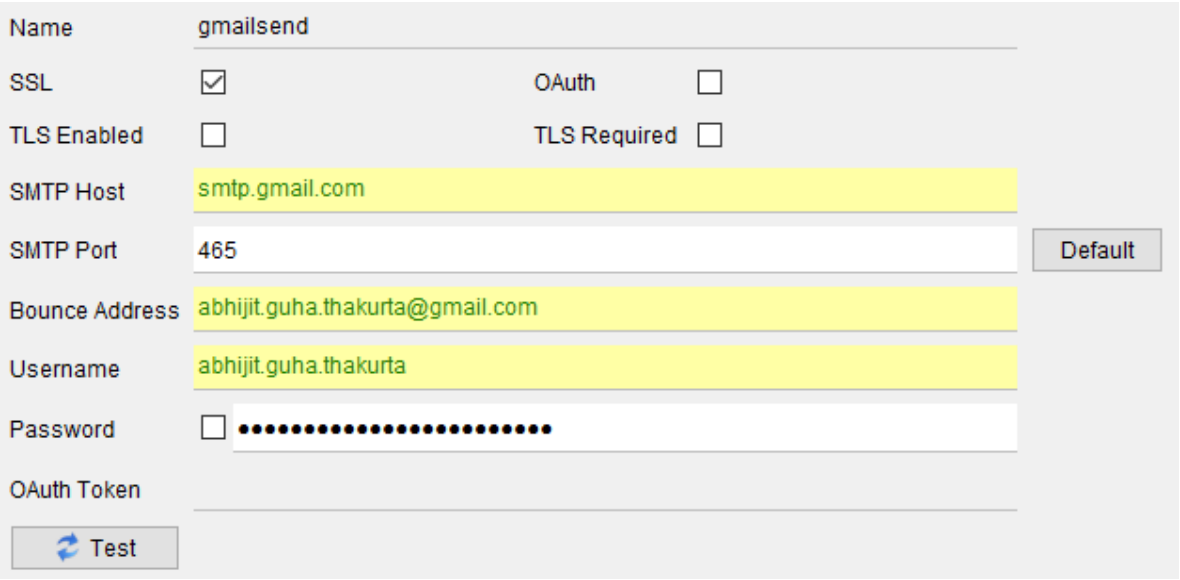


Figure 125 - SMTP Connector editor

PARE manages thread specific SMTP connections – which are automatically released once the concerned Thread has finished doing its tasks. Once this connector is defined, users can create 'Email Send' actions that uses this connector to send emails.

Name	Name of the SMTP Connector
SSL	Sets the 'SSL on connect' flag in SMTP properties
TLS Enabled	Sets the TLS enabled flag in SMTP properties
TLS Required	Sets the TLS required flag in SMTP properties
OAuth	In this case OAuth authentication (if the provider supports it) will be used instead of password. Corresponding token expression needs to be provided.
SMTP Host	Host name or IP address of the SMTP server
SMTP Port	Port name of the SMTP server that supports the selected encryption
Default	Default port – automatically set when one of the encryption setting is set or changed: • No encryption: port 25 • SSL on connect: port 465 • TLS enabled / required: port 587
Bounce Address	Sets a hint for the recipient server as to where the bounced email will be returned
Username	SMTP server username
Password Name	SMTP server password name (actual password defined in Password Manager)
Test	Connects to the SMTP server or shows an error.

3.10.4 POP3 / IMAP Connector

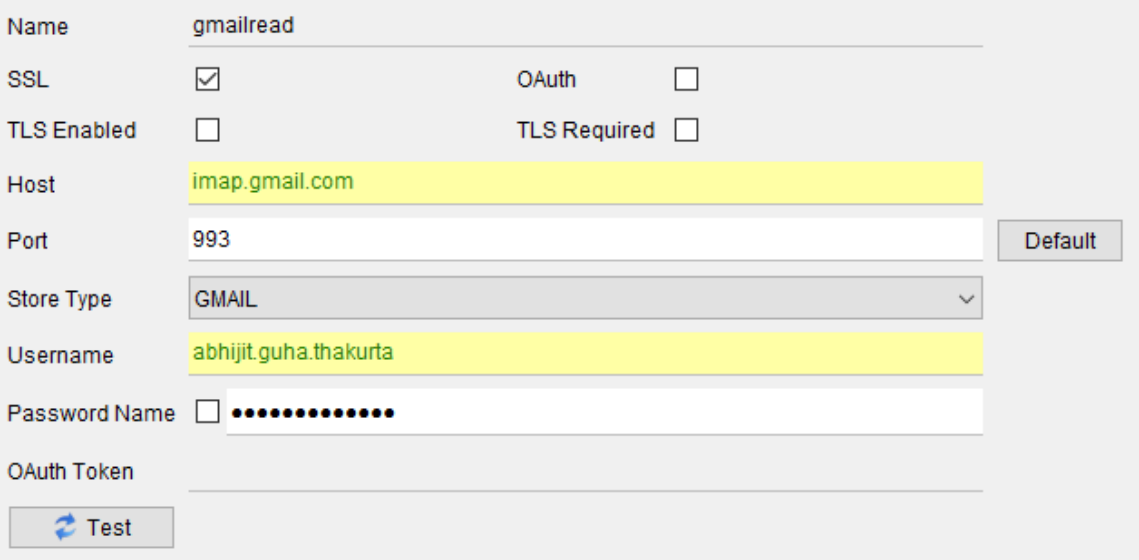


Figure 126 - POP3 / IMAP Connection editor

The POP3 / IMAP connector establishes a session with the email server to read emails. The POP-IMAP email Data Source uses this Connector to deliver email data.

Name	Name of the POP3/IMAP Connector
SSL	Sets the 'SSL on connect' flag in connector properties
TLS Enabled	Sets the TLS enabled flag in connector properties
TLS Required	Sets the TLS required flag in connector properties
OAuth	In this case OAuth authentication (if the provider supports it) will be used instead of password. Corresponding token expression needs to be provided.
Host	Host name or IP address of the email server
Port	Port name of the email server that supports the selected encryption
Default	Default port – automatically set when one of the encryption setting is set or changed: • No encryption: POP3-110 / IMAP-143 • SSL on connect: POP3-995 / IMAP-993 • TLS enabled / required: POP3-995 / IMAP-993
Store Type	Whether POP3, IMAP or GMAIL (GIMAP) protocol will be used for downloading emails
Username	Email server username
Password Name	Email server password name (actual password defined in Password Manager)
Test	Connects to the email server or shows an error.

3.10.5 FTP Connector

FTP connector manages connection to a FTP server. It supports following encryption protocols:

- Insecure
- TLS Explicit
- TLS Implicit
- SFTP (SSH)

Create the connector Node in standard way by right clicking on the Connectors folder.

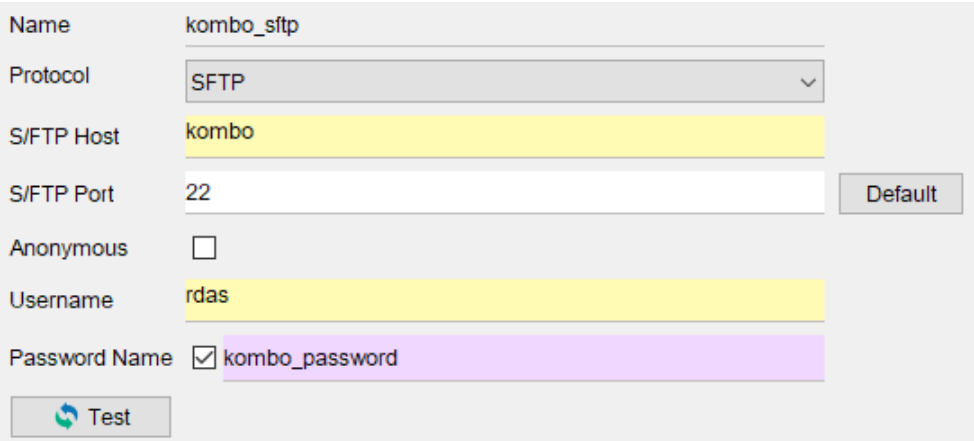


Figure 127 - FTP Connector editor

PARE manages thread specific FTP connections – which are automatically released once the concerned Thread has finished doing its tasks. It also maintains a background evictor thread that periodically checks and evicts stale connections. Once this connector is defined, users can create following PARE resources:

- Ftp Upload Action
- Ftp Download Action
- Ftp Delete Action
- Ftp List Function (returns the list of files)

Name	Name of the FTP Connector
Protocol	Any one of: • Insecure • TLS Explicit • TLS Implicit • SFTP (SSH)
FTP Host	Host name or IP address of the SMTP server
FTP Port	Port name of the SMTP server that supports the selected encryption
Default	Default port – automatically set when one of the encryption setting is set or changed: • Insecure: port 21 • TLS Explicit: port 21 • TLS Implicit: port 990 • SFTP (SSH): port 22
Anonymous	Sets Username as "Anonymous" and password as a dynamic string. Selecting it also disables the Username and Password fields.
Username	FTP server username
Password Name	FTP server password name (actual password defined in Password Manager)
Test	Connects to the SMTP server or shows an error.

3.10.6 Class Connector

The Class Connector helps load and use a Custom Java object. This is the underlying resource for exposing Custom Actions and User Functions in PARE. A Class Connector can load a Class (an Objects template) from either a JAR file or a directory. Once loaded, the Class Connector can create Objects from the class using the Constructor Parameters defined in the Connector.

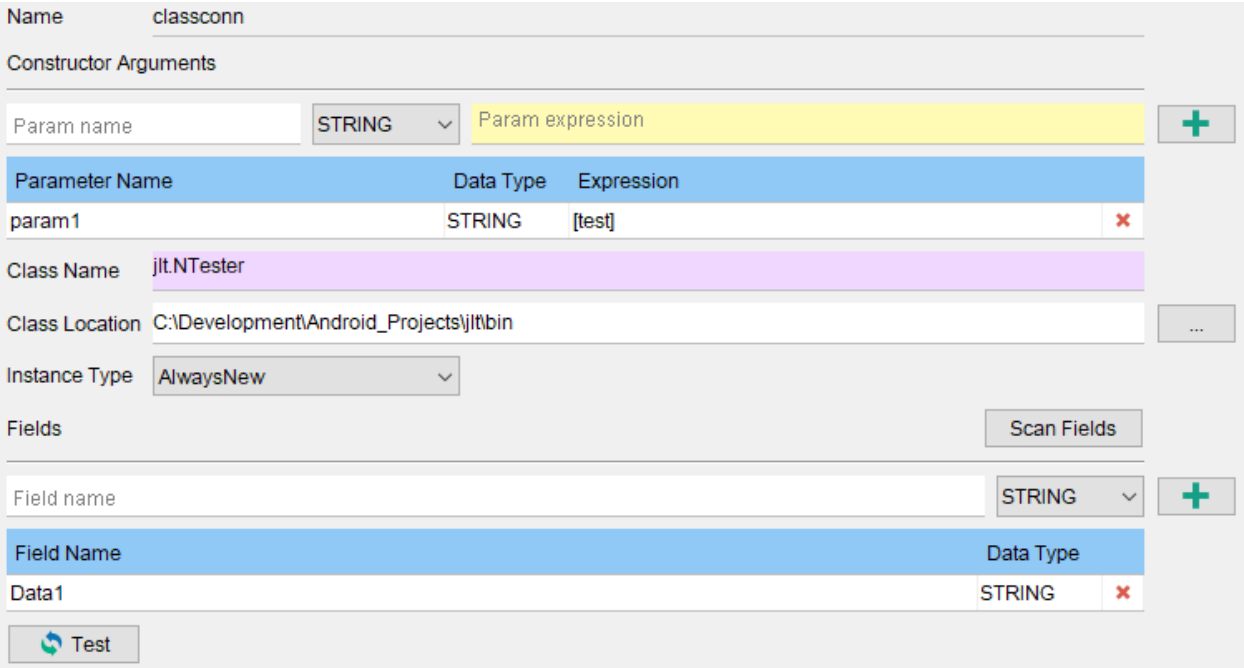


Figure 128 - Class Connector editor

Name	Name of the Class Connector
Constructor Arguments	The argument expressions that will be used when an Object instance of the Class is requested. A Class Connector creates a single Object per thread, on-demand. Each Constructor argument consists of: • Name of the argument • PARE DATATYPE of the argument • PARE Expression that provides value of the argument
Class name	Name of the Class that will be loaded on-demand. Note: Fully qualified class name needs to be provided. **Note**: once the Class Location is set, the designer automatically loads the Classes present on the location. Clicking Ctrl-SPACE on the Class Name field, brings up a popup menu with those names for selection by user.
Class Location	Path of the JAR file or directory which contains the class definition. Note: In case directory is provided, it should be the root folder, so that the class loader can traverse the directory thereafter based on the fully qualified class name.
Instance Type	• Singleton: Only ONE object instance of the class will be ever created as part of the PARE instance. • Per Thread: An object instance is created per thread. • Always New: A new Object Instance will be created per access.
Scan Fields	A Class Connector can search the loaded Class for Field candidates. A Field candidate must satisfy the following conditions: • It must have a corresponding PARE DATATYPE • A 'setter' method must be present • A 'getter' method must be present The 'Scan Fields' button searches the Class and automatically adds fields. Please **note** that this step only identifies the Field Candidates. To expose the Fields in PARE, users require to: • Create an Object Data Source using this Class Connector • Create DataSource Fields from the Object Data Source
Add Field	In some situations, the Class Connector may not have immediate access to the Class resources (JAR or directory). It therefore allows manual definition of the Fields.

Test	Creates an instance object of the Class or shows an error.

Class Connectors can be used to define Custom Actions and User Functions. It includes a Method parser accessible through right-click menu:

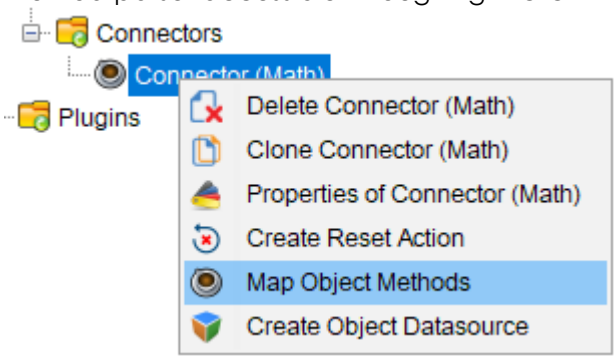


Figure 129 - Map Object assets

Selecting above option brings up the Method selection window. Class public methods are automatically categorized as Functions and Actions for user to select and auto-create the nodes.

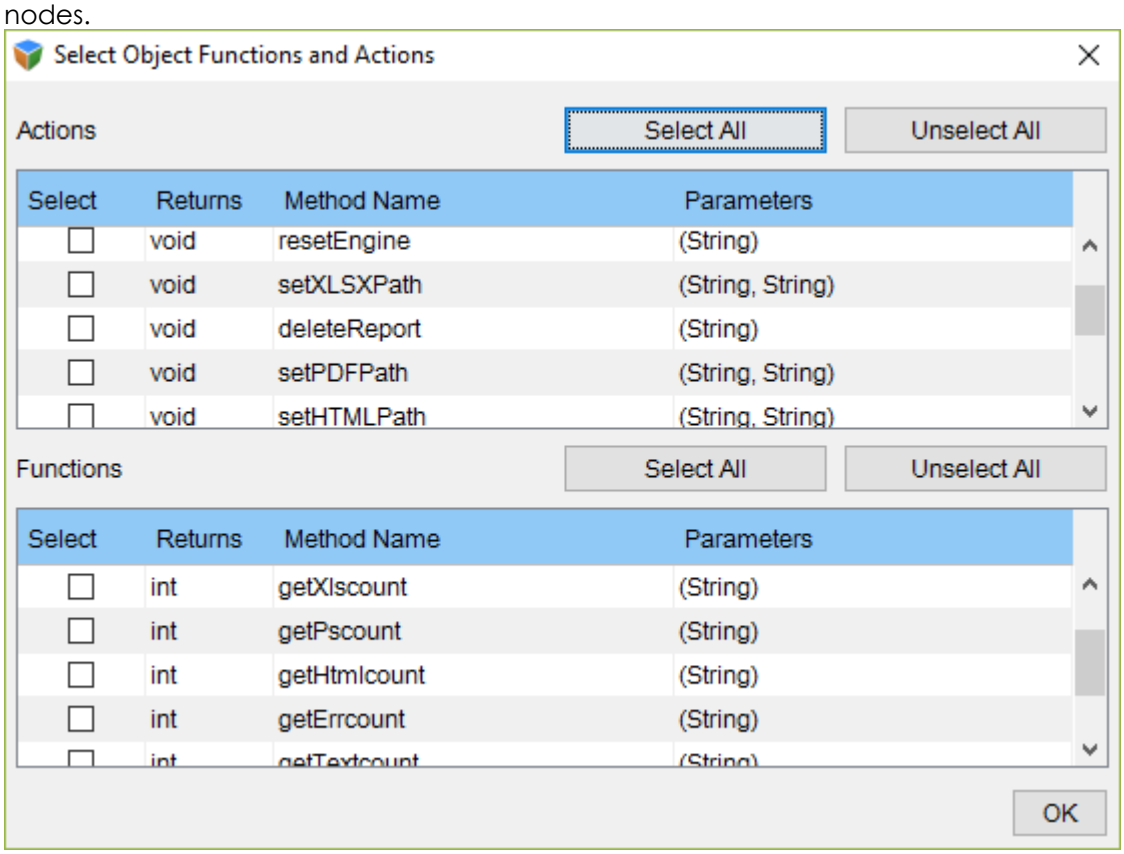


Figure 130 - Select Object Assets

3.10.7 Socket connector

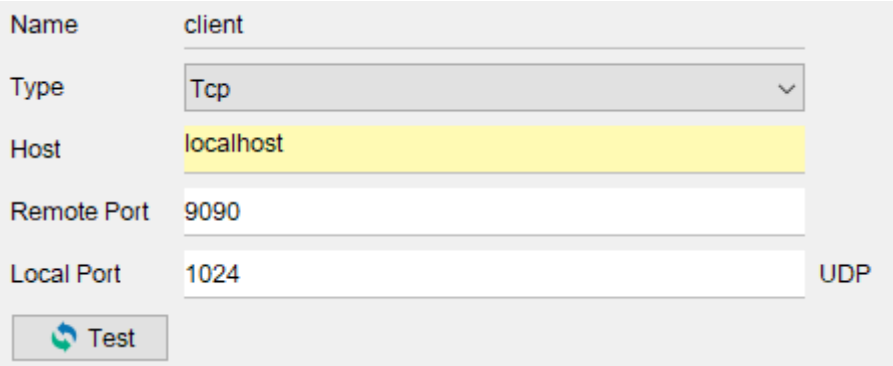


Figure 131 - Socket Connector editor

The Socket Connector provides capability to either:

- Connect to a remote TCP server and do data exchange (read and write)
- Send (write) UDP Packets to a remote UDP listener
- Receive (read) UDP Packets from a remote client

The Socket Read and Write actions depend on this Connector.

Name	Name of the Socket Connector
Type	Any one of: • TCP • UDP
Host	Remote hostname or IP address to connect in TCP mode, or send Datagrams to, in UDP mode.
Remote Port	Remote port to connect in TCP mode, or send Datagrams to, in UDP mode.
Local Port	The local port number on which the UDP read will happen. Remote clients sending Datagrams via UDP should sending to this port.
Test	Tests the connector by making dummy connections.

3.10.8 Server Socket Connector

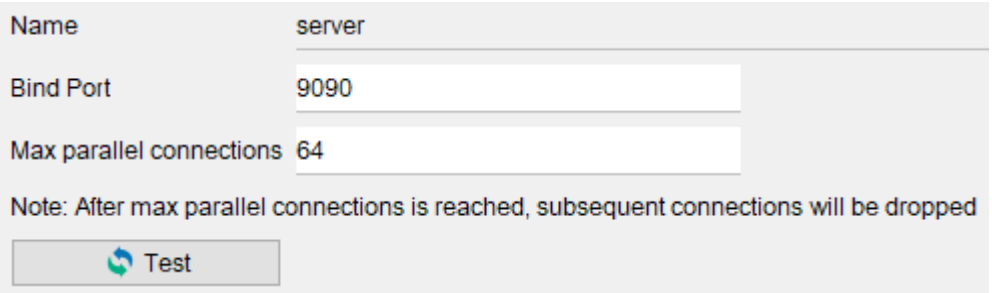


Figure 132 - Server Socket editor

The Server Socket Connector configures a TCP Server Socket.

Name	Name of the Server Socket Connector
Bind Port	The local port where the TCP server will listen. Remote TCP clients will need to connect to this port.
Max Parallel Connections	With PARE TCP Server users can limit the maximum number of active connections that can be operated on simultaneously. Once the number of active connections to this Server exceeds this limit, all new Connection attempts are rejected. The hard limit to the number of parallel Connections is currently set at 4096.

Test	Test the Connector by creating a dummy connection.

3.10.9 SAP Connector

SAP Connector is used to establish a JCO connection to an existing SAP R3 instance. This Connector is used onwards to create the BAPI Functions as described here.

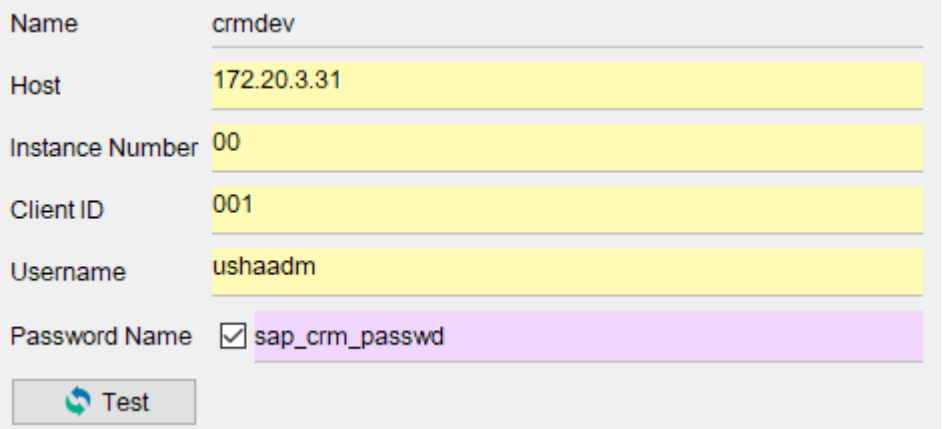


Figure 133 - SAP Connector editor

Name	Name of the SAP Connector
Host	Hostname or IP address of the node where SAP R3 instance is installed.
Instance Number	SAP Instance number as configured in the host system.
Client ID	JCo Client identifier
Username	SAP Login username
Password name	SAP Login Password name (actual password retrieved from Password Manager)
Test	Test the Connector by creating a JCo destination.

3.10.10 Mongo DB Connector

Figure 134 - Mongo DB Connection editor

Mongo DB Connector connects to Mongo DB. Currently this connector supports the Mongo Datasource that is used to extract data from the various Mongo DB collections.

Name	Name of the Connector
Connection String	Connection URL in MongoDB format
Username	Mongo DB Login username
Password	Mongo DB password
Auth Source	Mongo DB database that defines the Username
Test	Test the Connector by doing a 'ping' (no-op) on the DB

3.10.11 SOAP Connector

SOAP connector serves two purposes:

- Parses a provided WSDL URL and helps configure SOAP calls
- Provides underlying capability to call a SOAP API to a particular SOAP End Point

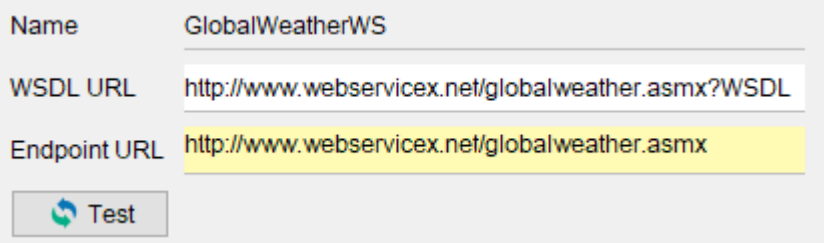


Figure 135 - Soap Connector editor

Name	Name of the SOAP Connector
WSDL URL	WSDL URL that provides definition of all the SOAP call signatures including definition of the input and output envelops
Endpoint URL	This expression helps determine the end point to be used to call the SOAP API. Note that the WSDL URL is required only when defining the SOAP Functions. Thereafter, for operations, only Endpoint URL is required.
Test	Test the Connector – checks access to the WSDL only

3.11 Data Sources

PARE can integrate with multiple data sources to expose all data that is required to make business decisions. It has capability, for example, to compare a value in a File with a value in a Database and take a decision, in a standardized way, without writing a single line of code, which would otherwise require a bit of involved work by the IT department.

This section delves deeper into the data sources supported by PARE.

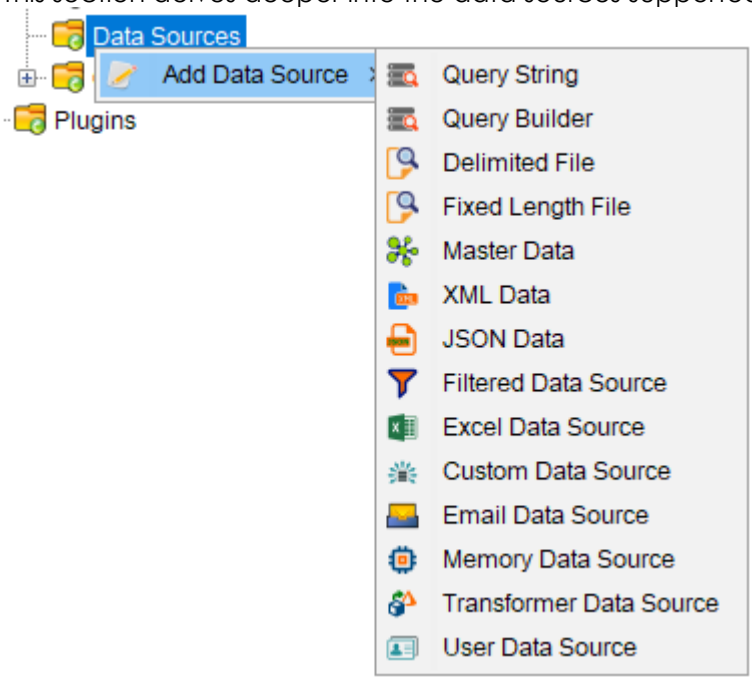


Figure 136 – Data Sources tree

3.11.1 Object Data Source

Users can create Object Data Source by Right clicking on a Class Connector:

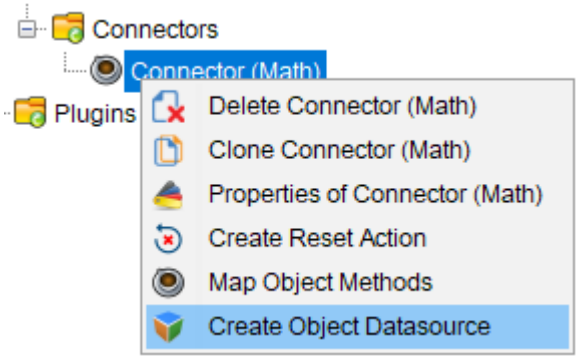


Figure 137 – Add Object Data Source

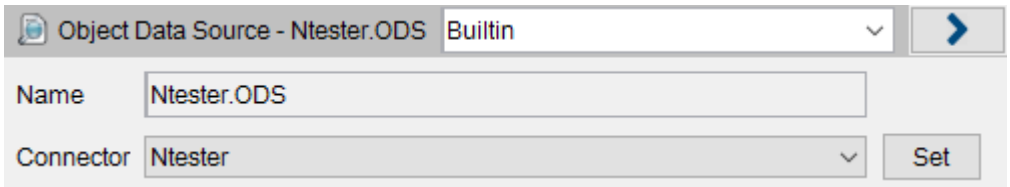


Figure 138 – Object Data Source editor

An Object Data Source is required in scenarios when a user needs to inject Application data into PARE for further processing. For example, the application is managing a list of objects and needs to process Rules on each of them. Then the user will take following steps:

Configuration steps:

- Define a Class Connector matching the Application Object
- Define an Object Data Source (ODS) based on the Class Connector
- Define a Job to iterate the ODS
- Define the Rules and other Actions to embed within the Job

Runtime steps:

- Load the ODS and use the 'addObjectInstance' API to inject the application objects
- Fire the ODS Job

Name	Name of the Data Source
Connector	The backing Class Connector

3.11.2 DB Query

DB Queries are SQL statements that are executed through a DB Connector. There are 2 variants of SELECT queries (Query String and Query Builder).

3.11.2.1 Query Builder

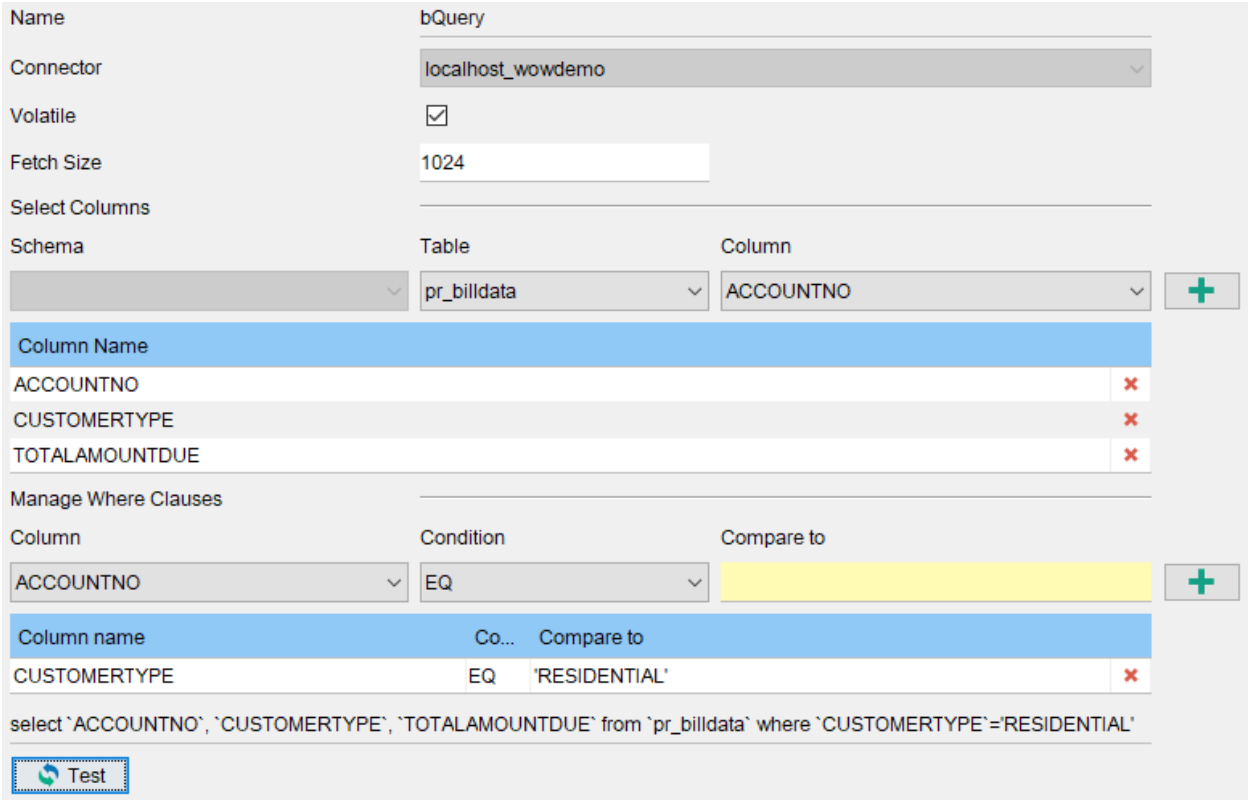


Figure 139 – Query Builder

The Query Builder makes it easy to write SELECT queries by exposing the table structure of the DB in the GUI. Users can select Scheme->Table->Column and add the Columns in the query. WHERE clauses can be included also where expressions can be embedded.

Name	Name of the Data Source
Connector	The Db Connector that will be used by this query
Select Columns	The Query Builder allows scanning of the DB Tables in the entire database starting from schema -> table -> column. Please note that, columns from only one table should be selected and added. Users should also make sure to include valid SQL data types (Numeric, String, Boolean, …).
Manage Where Clauses	Using Query Builder, 'where' clauses can be dynamically added or subtracted. All the columns of the selected table can be used to define a 'where' clause. PARE expressions can be used to define 'where' clauses.
Shared Connection	This flag indicates whether per thread connections will be created or a single connection from the pool will be shared to all calling threads. Typically, if the client threads are doing longer tasks using this query, this flag should not be checked.
Test	Executes the query. It does not show the results though. This step basically validates syntactical correctness of the query and ensures that the underlying connections are working fine.

Note:

- Relatively simple queries can be formed using the Query Builder, such as the select query can be written on only ONE table, Table joins are not possible, or AS clauses cannot be used and so on. If complex queries are required, Query String needs to be used.

- Query Builder does many background queries to the underlying DB to get all the meta-information. So, on big databases, the GUI response times can slow down when changing Schemas for example.

3.11.2.2 Query String

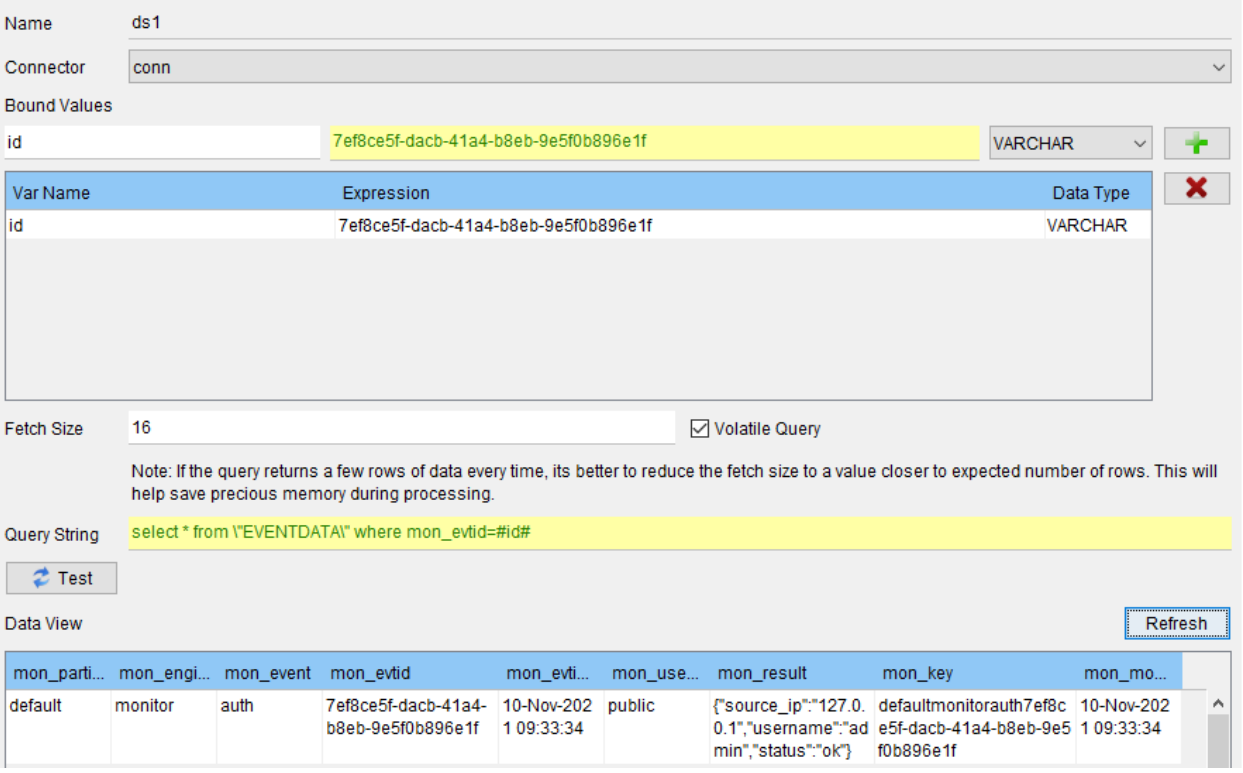


Figure 140 – Query String editor

Query String allows user to write a SQL query manually. Expression parser is available so that [Fields] and {Functions} can be embedded within the statement.

Query String now allows "bound values". Instead of directly embedding expressions within the query, they can be declared as bound variables and embedded within the query as #<var>#. Using bound values effectively mitigates the risk of SQL Injection attacks. The #<var># pattern provides a very flexible way of placing bound vars in the query itself.

Name	Name of the Data Source
Connector	The Db Connector that will be used by this query
Bound Values	Configure name and value expressions of the "bound variables" to be used in this query. All expressions that may cause a SQL injection attack should be converted to bounds vars here.
Fetch Size	Passes a hint to the driver to fetch these many rows at a time. This value affects memory consumption as well as performance. So, value of the same should be tuned to user requirements. Typically for queries returning few or single row, a big fetch size is not useful. But for queries that returns a lot of rows, a bigger fetch size can improve performance – at the cost of increased memory consumption.
Query String	You can enter queries here – as complex as may be possible. Since PARE expression can be used for whole query, this is the most flexible approach.
Dependencies	Shows other fields or functions that are referred in the expressions used in this Field.
Shared Connection	This flag indicates whether per thread connections will be created or a

	single connection from the pool will be shared to all calling threads. Typically, if the client threads are doing longer tasks using this query, this flag should not be checked.
Test	Executes the query. It does not show the results though. This step basically validates syntactical correctness of the query and ensures that the underlying connections are working fine.

3.11.3 File Data

PARE allows reading data from Files and mapping Fields using that data. Currently 2 different file formats are supported:

- Delimited record files (CR/LF terminated)
- Fixed length record files (CR/LF terminated)

3.11.3.1 Delimited File

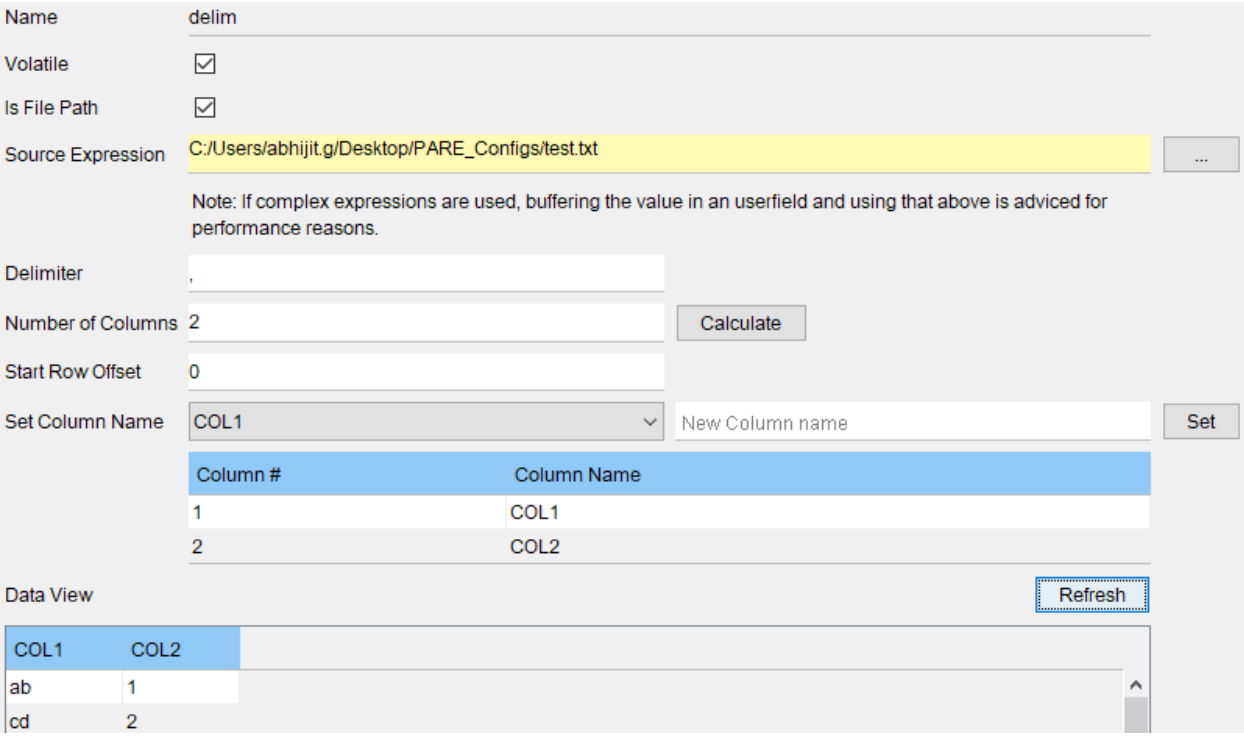


Figure 141 – Delimited File editor

The most common example of a delimited file is a CSV file that can be produced from MS Excel. There are similar file formats that use '|' instead of comma. PARE supports any delimiter and multi-character delimiters. The GUI can auto-calculate number of delimited columns by scanning the file, and the bottom table shows data view – first 20 lines of the file. Column names are generated automatically and users are allowed to set meaningful names per column.

Name	Name of the Data Source
Is File Path	If set to False, the File Path expression is treated as the actual content of the file. Else, the its treated as file path and the content read from there.
File Path	Path of the file. PARE expression used to allow dynamic file names.
Dependencies	Shows other fields or functions that are referred in the expressions used in this Field.
Delimiter	Delimiter to be used. For CSV files, for example, this should be ",".
Number of Columns	For delimited files user can click calculate to auto-populate this field (recommended). Otherwise it can be manually entered also. Number of columns greater than calculated amount may result in error during processing.

Start Offset	Skip a pre-set number of lines from the start of the file (typically header rows).
Set Column Name	Since column names are auto-populated when number of columns is entered, this option allows over-riding those names with more meaningful application specific names (like start-time, b_number etc).
Data View	This shows the result of the configuration dynamically. Only first 20 rows of the file are shown.

3.11.3.2 Fixed Length File

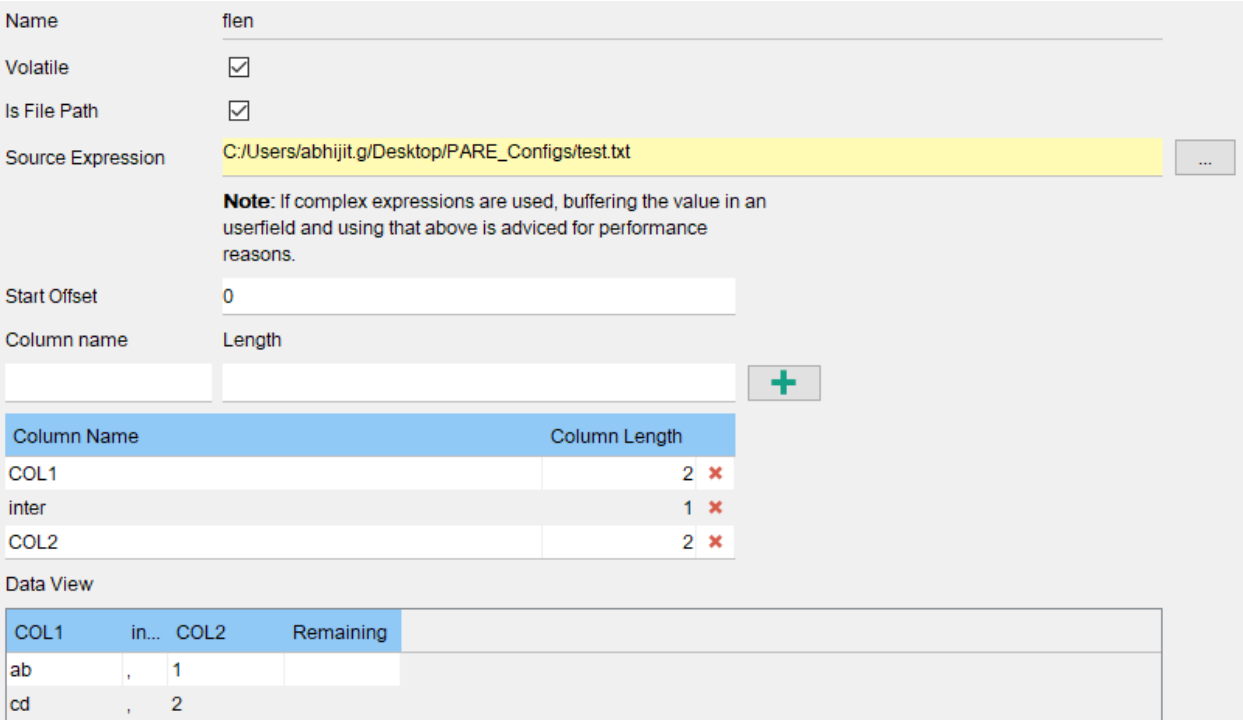


Figure 142 – Fixed Length File editor

In Fixed Length scenario, Columns are organized as follows:
[col1][col2][col3][col4]…

Each column starts from the end of the previous column and the first column starts at the beginning of the record. Hence, while adding columns, only length of the column needs to be provided. The GUI helpfully shows both column-mapped data plus the remaining un-mapped data so that when configuring the columns, immediate visual feedback is available.

Name	Name of the Data Source
File Path	Path of the file. PARE expression used to allow dynamic file names.
Dependencies	Shows other fields or functions that are referred in the expressions used in this Field.
Start Offset	Skip a pre-set number of lines from the start of the file (typically header rows).
Columns	In FL files, Columns are always manually defined and a length needs to be input per column.
Data View	This shows the result of the configuration dynamically. Only first 20 rows of the file are shown.

Note: Configuring delimited files is easier – for obvious reasons. So if delimiters exist within the source file, that method is recommended.

3.11.4 Master Data

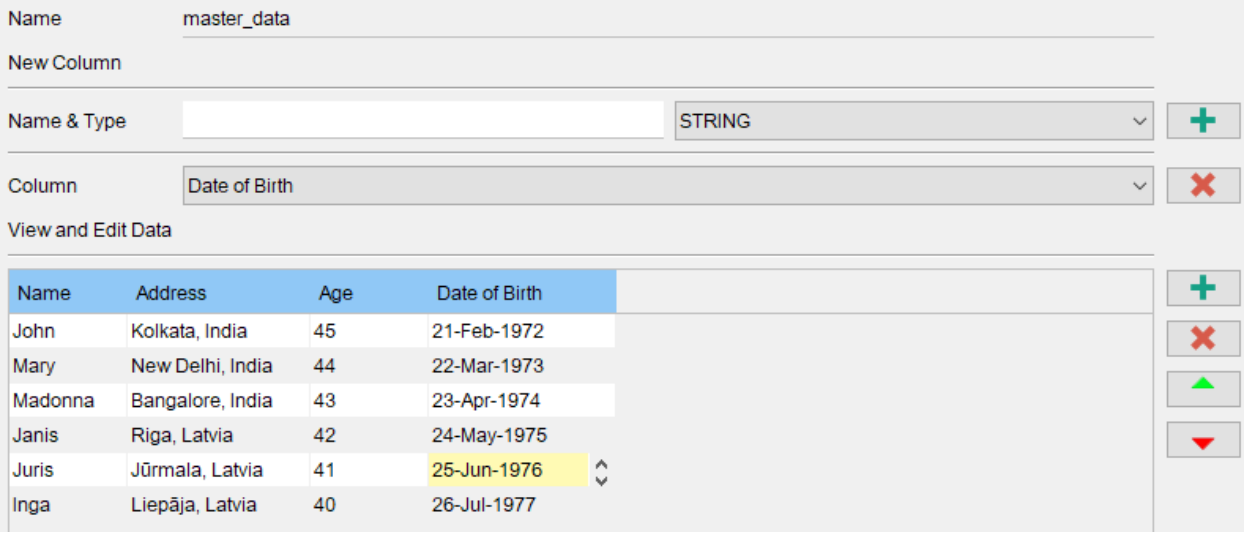


Figure 143 – Master Data editor

Master Data editor allows creation of simple Tabular Data that are stored as part of the Rules XML. All PARE supported Data Types are allowed. Master Data node can be used as a Data Source and the data can be both read and edited at run time by PARE Actions.

Name	Name of the Data Source
Columns	Add named columns to the Data Source
Set Data	Select an existing column from the drop down and enter the value as String. The corresponding cell (intersect of the selected column and current selected row in the table) will be updated.
Data View	Shows the data in tabular format. Also allows addition and deletion of rows.

3.11.5 XML Data Source

XML DataSource can parse any XML file (or XML string) and present its data in a tabular format.

XML Data Source uses XPaths to pin-point an Element or Attribute node and extract value at run time from a given XML file. The Data Source editor includes XML structure browser that can be used to select nodes to be exposed.

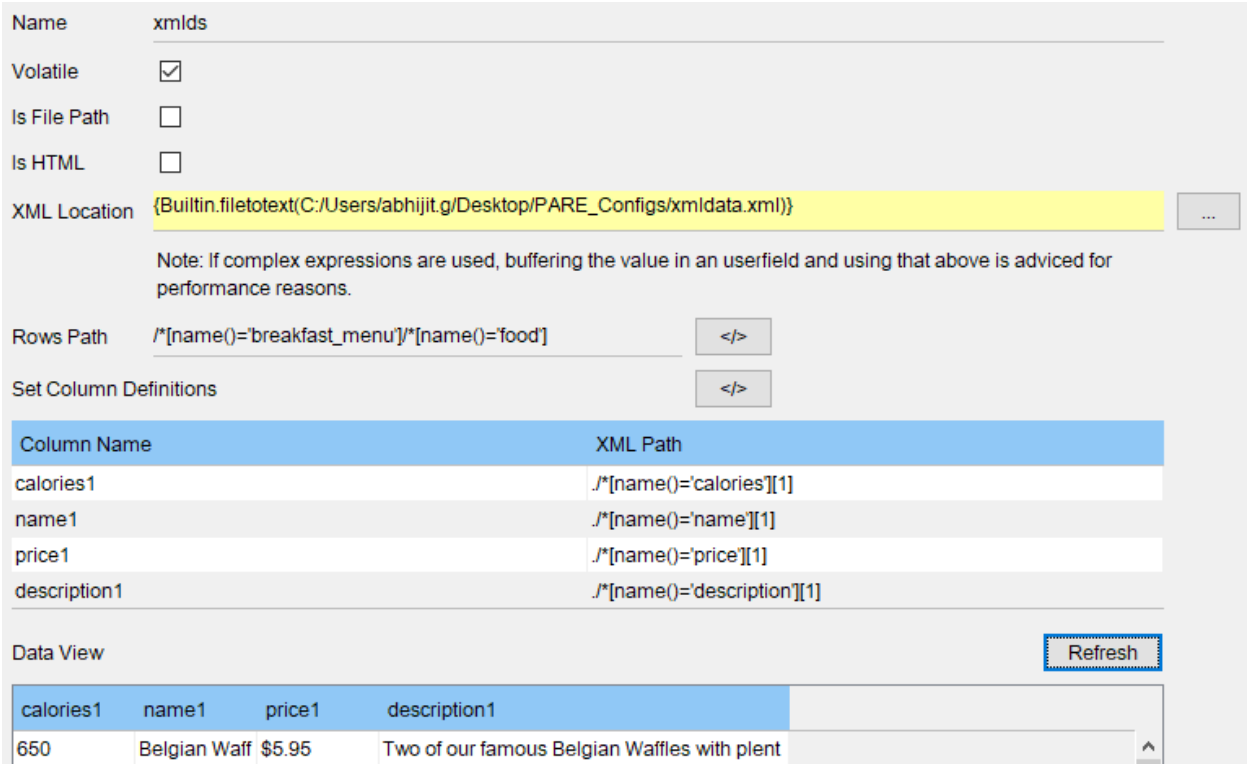


Figure 144 - XML Data Source editor

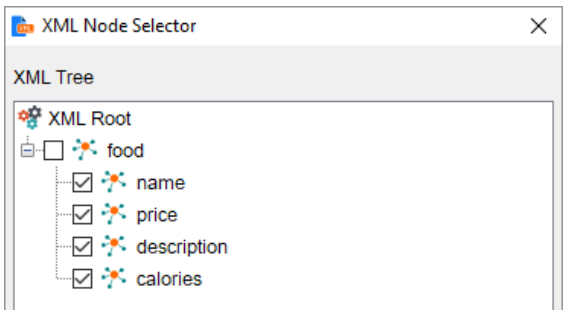


Figure 145 - XML Node browser

Name	Name of the Data Source
Is File Path	Either a XML filename or XML content itself can be given as input to the Data Source. This flag determines if the Input will be interpreted as a File or content string.
Is HTML	The XML Data Source can double up as HTML Parser as well. If this flag is checked, the content (file or passed content) is evaluated as HTML instead of XML.
XML Location	Expression that either indicates a XML File location or the XML content itself, based on the "Is File Path" flag.
Rows Path	The Rows Path button invokes the Node Browser and let's select an Element. If multiple Elements with same node-name exists in the same hierarchy level, each such element is considered a separate Row of the Data Source. **Note:** - Only an Element node can be selected as a Row. - Only ONE Element can be selected in the hierarchy.
Column Definitions	This button invokes the Node Browser again but shows the hierarchy below the first selected Row. In this scenario, both Elements and Attribute nodes can be selected. The Column definitions include an auto-generated column name and relative path of the selected Element or

	Attribute from the corresponding Row Node.

3.11.6 JSON Data Source

JSON Data Source can parse any given JSON String (or a file containing the JSON string) and present its contained data in a tabular format.

JSON Data Source uses Jackson style JSON Pointers to pin-point a data element and extract value at run time. The Data Source editor includes JSON structure browser that can be used to select nodes to be exposed.

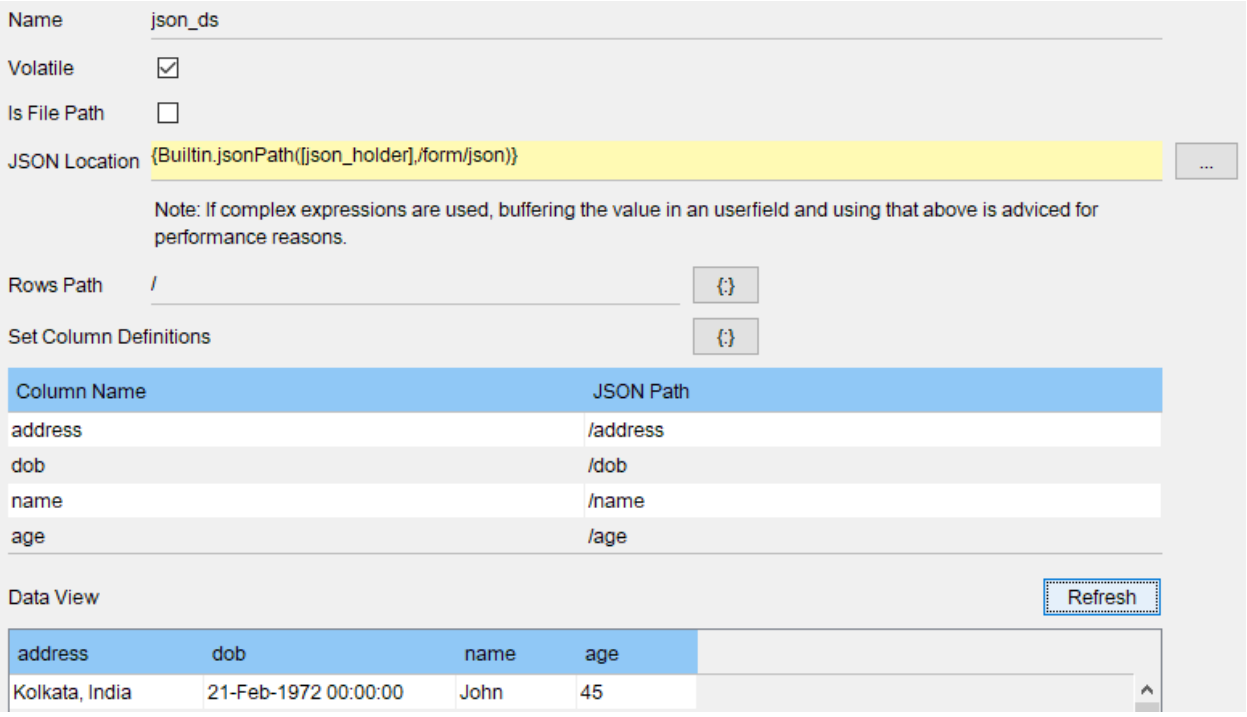


Figure 146 - JSON Data Source editor

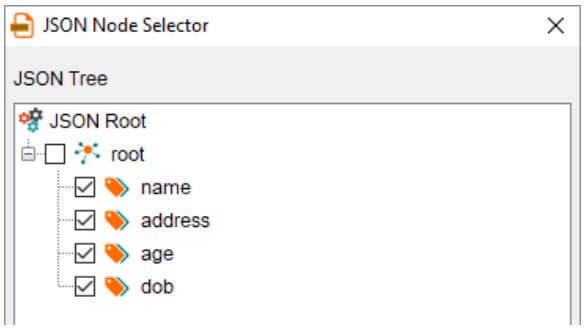


Figure 147 - JSON Node Selection editor

Name	Name of the Data Source
Is File Path	Either a JSON filename or JSON String itself can be given as input to the Data Source. This flag determines if the Input will be interpreted as a File or content string.
XML Location	Expression that either indicates a JSON File location or the JSON content itself, based on the "Is File Path" flag.
Rows Path	The Rows Path button invokes the Node Browser and let's select a JSON node. Only ARRAY [] type nodes can be selected as ROW nodes. **Note**: This Data Source is geared towards extracting tabular data, and hence the Row node must be of Array form. If user is interested to extract a single value from the JSON, either of 'parseJson' or 'jsonPath' Builtin

	function can be used.
Column Definitions	This button invokes the Node Browser again but shows the hierarchy below the first selected Row. In this scenario, both ARRAY and normal nodes can be selected. The Column definitions include an auto-generated column name and relative path of the selected Node from the corresponding Row Node.

3.11.7 Email Data Source

Email Data Source allows reading an IMAP/GMAIL/POP3 mailbox and expose mail data as a tabular Data Source. Users can connect to the mailbox using POP3/IMAP/GMAIL connector and filter the emails that needs to be exposed. To create an Email Data Source, use the right-click options menu.

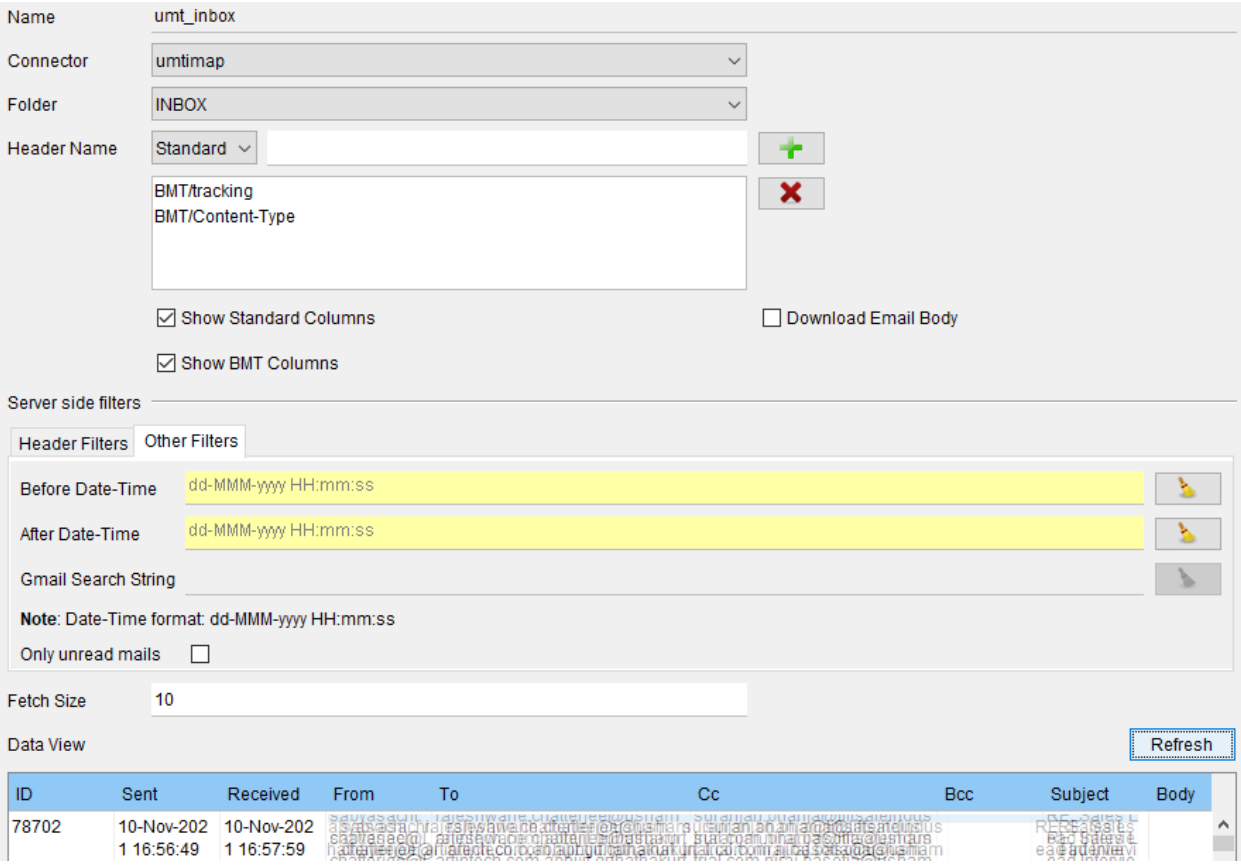


Figure 148 - Email Data Source editor

Name	Name of the Data Source
Connector	The POP3/IMAP connector (the email store) used to render this Data Source
Folder	The email folder from which emails will be downloaded
Header Name	By default, following headers are downloaded: • Id: A unique email ID • Sent: Sent timestamp • Received: Received timestamp • From: Received from email address • To: Sent to email address(s) • Cc: Cc'd to email address(s) • Bcc: If mail is received as bcc • Subject: Email subject • Body: Optionally body can be downloaded

	Apart from this, if any other email header is required to be exposed, those header names can be added here.
Download email body	This flag enables download of the email body. This is optional as this can be a time-consuming activity once enabled.
Filters	Email Data Source supports following filters: • Header filters: Filtration by presence or absence of a String one or more of the configured headers • Before and After: Filtration by time box • Unread only: Whether only unread mails will be downloaded • Gmail Search: This is applicable if the underlying Connector is using the G-IMAP protocol (for Gmail Inboxes). User can enter a standard search string just like they do in Gmail web interface.
Fetch size	The fetch of emails happens in buckets. The fetch size sets the size of the bucket and controls how much information is downloaded in a single trip to the email server.
Data View	Shows the data in tabular format. Also allows addition and deletion of rows.

3.11.8 User Data Source

User Data Source (UDS) allows dynamic data creation capability within PARE. Users just need to configure the columns (and corresponding PARE Data Types) in the editor. UDS allows configuration of Actions to add or delete rows within the data source. The Data Source Field setters also update the UDS values at the current cursor.

User Data Sources are created by right-clicking on the Data Sources tree node.

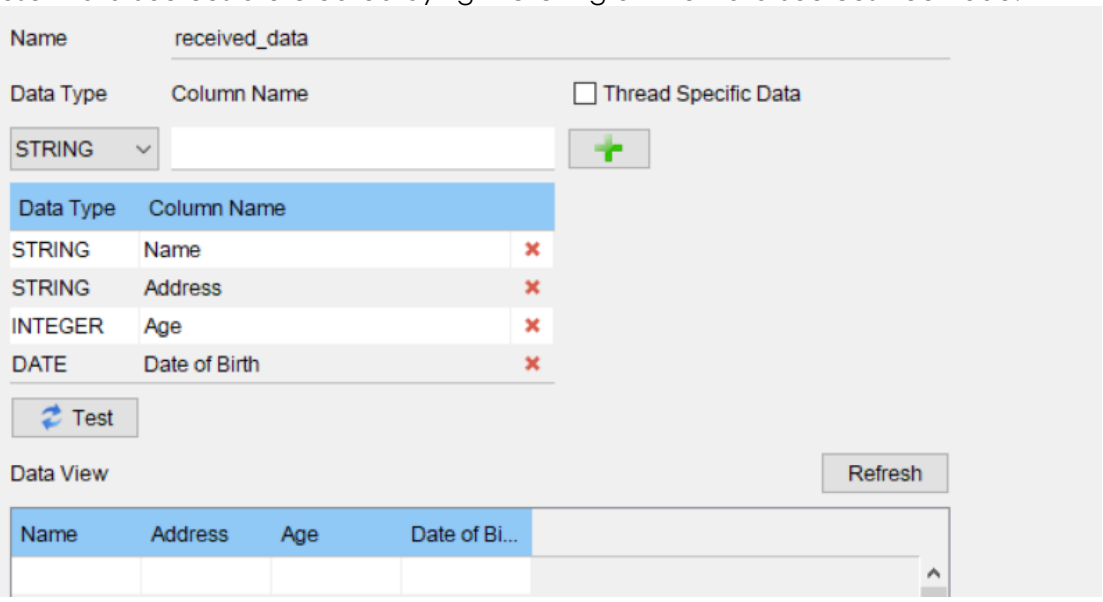


Figure 149 - User Data Source editor

User applications can assign values to the UDS by calling the 'addRow' API method. UDS is also the only data source that can be used to consume JSON or XML data via Models.

Name	Name of the Data Source
Column Name	Name of the UDS column
Data Type	PARE Data type of Column
Thread Specific Data	The data loaded into the UDS says Thread Specific – so it expires with expiry of the calling thread.

This can be used with a Job. Corresponding Data Source Field can be created by right clicking on the List Data Source node.

3.11.9 Excel Data Source

Excel Data Source can read data directly off Microsoft XLS or XLSX files. It uses the Apache POI library for reading OOXML (XLSX) and OLE2 (XLS) file formats. Excel data source can be used to map tabular data inside a sheet of an Excel workbook as follows:

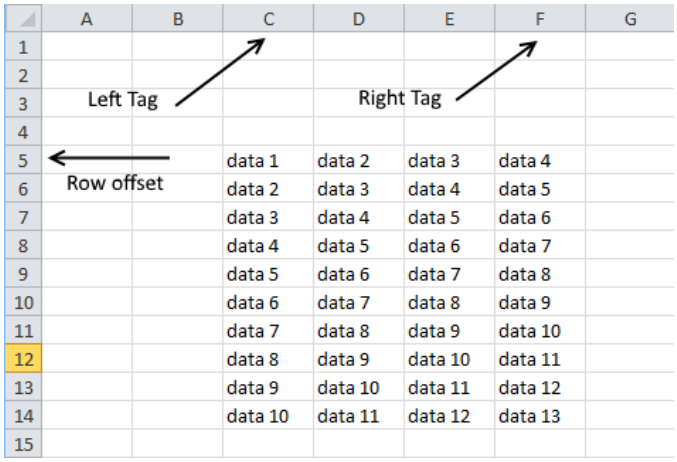


Figure 150 - Excel sample data

The Excel file is an expression. Rest of the configurations help define an area of the Sheet that needs to be mapped as a PARE Data Source.

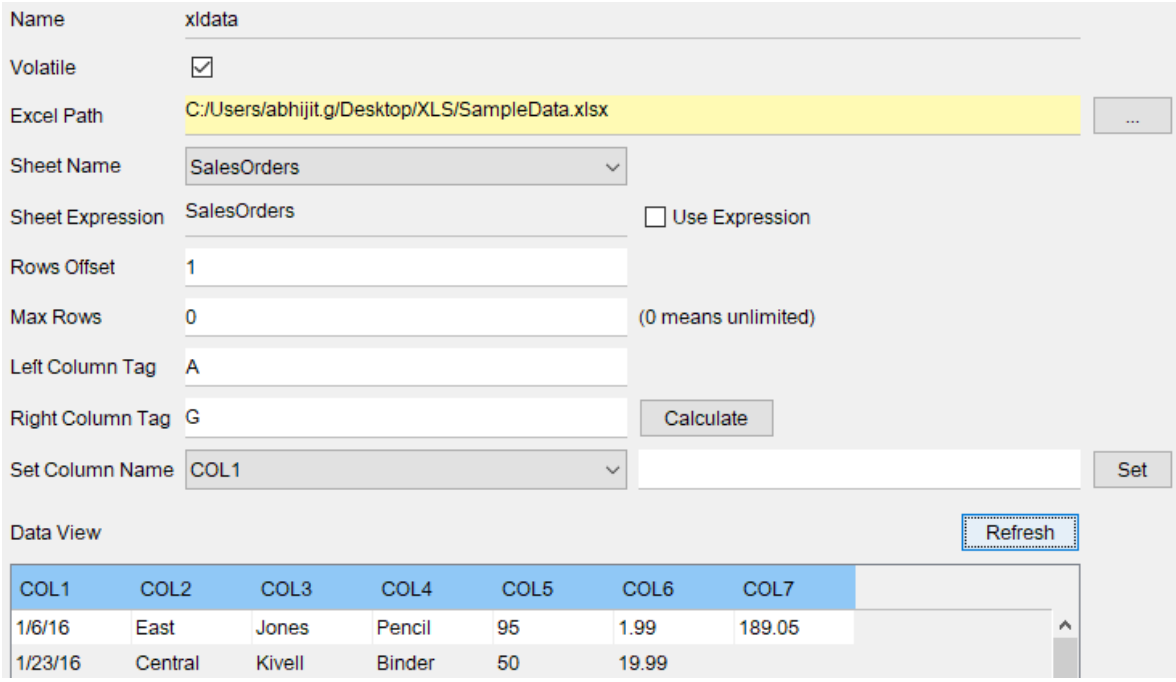


Figure 151 - Excel Data Source editor

Name	Name of the Data Source
Excel Path	Path of the XLS or XLSX file. PARE expression used to allow dynamic file names.
Dependencies	Shows other fields or functions that are referred in the expressions used in this Field.
Sheet Name	Name of the Sheet from where data is to be read. On providing a valid input filename, the sheet names are automatically loaded in the

	dropdown for easy selection by user.
Sheet Expression	Sheet names can be optionally provided using PARE Expression syntax. This makes the sheet name dynamic.
Rows offset	Number of rows to skip from the top of the selected sheet.
Max Rows	Maximum number of rows to read after the offset. If this is set to ZERO, reading will continue until an empty row is encountered. Note: An empty row marks the end of the Data Source in all cases.
Left Column Tag	Left column tag which should be treated as first column of the Data Source
Right Column Tag	Right column tag which should be treated as the last column of the Data Source
Calculate	This helper auto-populates the Left and Right column tags based on data present on the Offset row position of the selected sheet. Note: Its recommended that users populate the Left and Right tags manually after checking actual data in the excel sheet.
Set Column Name	Since column names are auto-populated when number of columns is entered, this option allows over-riding those names with more meaningful application specific names (like start-time, b_number etc).
Data View	This shows the result of the configuration dynamically. Only first 20 rows of the file are shown.

3.11.10 Mongo DB Data Source

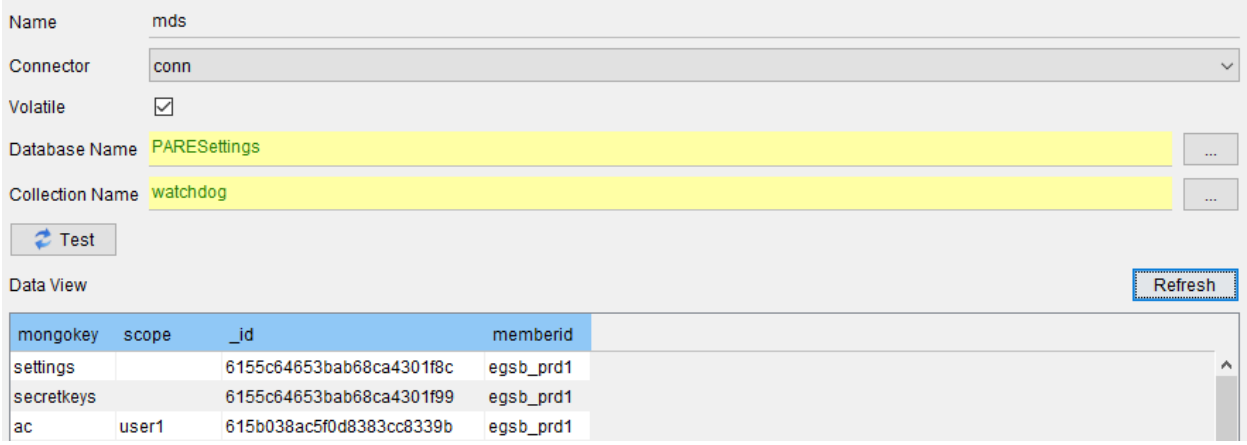


Figure 152 - Mongo DB Data source editor

The Mongo DB Data Source allows reading of a collection. The Data Source automatically retrieves all columns (union of fields within the records of the selected collection).

Name	Name of the Data Source
Connector	Mongo DB Connector to be used by the Data Source
Volatile	If checked, the DB Name and Collection names are re-evaluated as every access. Otherwise, they are evaluated once at the start and thereafter whenever the 'refresh' action is called on the Data Source.
Database Name	Name of the Mongo DB database
Collection Name	Name of the Collection within the database
Data View	This shows the result of the configuration dynamically. Only first 20 rows of the file are shown.

3.11.11 Filtered Data Source

Filtered Data-source applies a condition filter on an existing Data-source. It's useful to attach filtration conditions on an existing data source where no such facility (like SQL WHERE Clauses) is already available, e.g. File Data-source, Object Data-source etc.

Note: Filtered Data-source auto-advances the cursor till the condition is satisfied.

It's created by right-clicking on the Data Sources node.

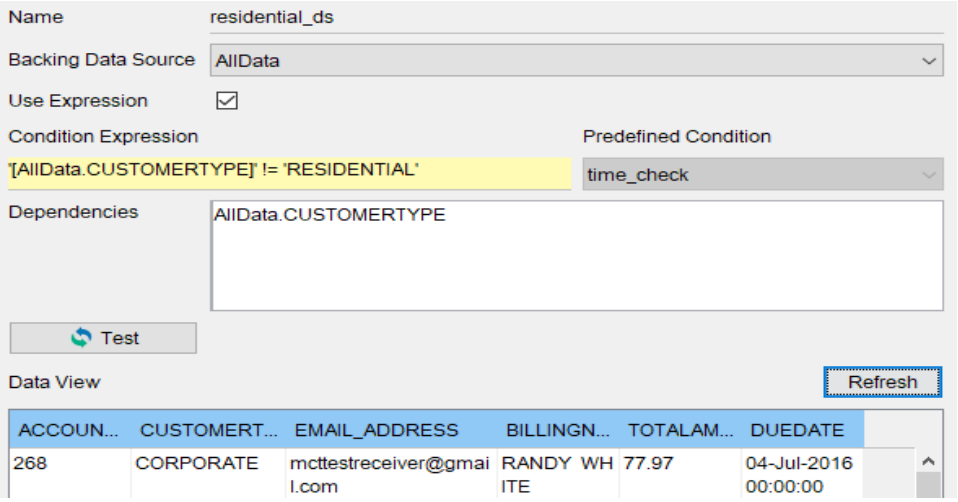


Figure 153 - Filtered Data-source editor

Name	Name of the Data Source
Backing Data Source	An existing Data Source on which the Filtration will be applied
Use Expression	Flag to determine whether a pre-defined condition or a condition expression will be used as filtration
Condition Expression	Condition expression
Pre-defined Condition	Pre-defined condition
Test	Test button tests the Data Source and returns any error that might be received

3.11.12 Custom Data Source

Any type of custom structured data can be exposed in PARE using the Custom Data Source. PARE exposes a Java interface that a Custom Data Source must implement. The resulting classes can be loaded in PARE using its class loader and PARE automatically configures a Custom Data Source from it.

The interface that needs to be implemented:

```
/**
 * TabularData interface defines the template which can be used to
 * create custom data sources that are usable by PARE. For example,
 * this can be used to parse custom spool files or XML or JSON data
 * sources, which PARE built-in DataSources cannot directly handle.
 * <br><br>
 * About thread safety:<br>
 * PARE dynamically creates a new instance of the TabularData object per
 * thread. So, using instance variables is safe. However, if static
 * variables are used, access to those must be synchronized.
 * <br><br>
* @author abhijit
 *
 */
```

```
public interface TabularData {

	/**
	 * Return the column names
	 * @return List of Strings (column names)
	 * @throws Exception
	 */
	public List<String> getColumnNames() throws Exception;

	/**
	 * Advance the cursor and return True. If already at the last row, return
	 * false.<br>
	 * Note: TabularData assumes that the implementing object maintains
	 * the current cursor. When the data source is first created, the cursor
	 * position just before the first row of data, so that calling 'next'
	 * once brings it to the first row.
	 * @return true or false (false, if the data source has reached the end)
	 * @throws Exception
	 */
	public boolean next() throws Exception;

	/**
	 * Returns true if the cursor is already before first row
	 * @return
	 * @throws Exception
	 */
	public boolean isBeforeFirst() throws Exception;

	/**
	 * Moves the cursor to just before first row - this is default
	 * position when the data source is opened
	 * @throws Exception
	 */
	public void beforeFirst() throws Exception;

	/**
	 * Get the column DATATYPE of the data source
	 * @param colname
	 * @return DATATYPE
	 * @throws Exception
	 */
	public DATATYPE getColumnDataType(String colname) throws Exception;

	/**
	 * Returns the value for the named column at the row indicated by current
	 * cursor.
	 * @param colname
	 * @return Value object<br>
	 * Return type should correspond to DATATYPE:<br>
	 * STRING - String<br>
	 * BOOLEAN - Boolean or boolean<br>
	 * DATE - java.util.Date<br>
	 * TIME - java.util.Date<br>
	 * DATETIME - java.util.Date<br>
	 * DOUBLE - Double or double<br>
	 * FLOAT - Float or float<br>
	 * INTEGER - Integer or int<br>
	 * LONG - Long or long<br>
	 * @throws Exception
	 */
```

```
        public Object getColumnValue(String colname) throws Exception;

        /**
         * Release any resources associated with the object. PARE will call this
         * method when the thread that was using this object is no longer active.
         */
        public void close();

}
```

Create the Custom DataSource node by right clicking on the Data Sources folder.

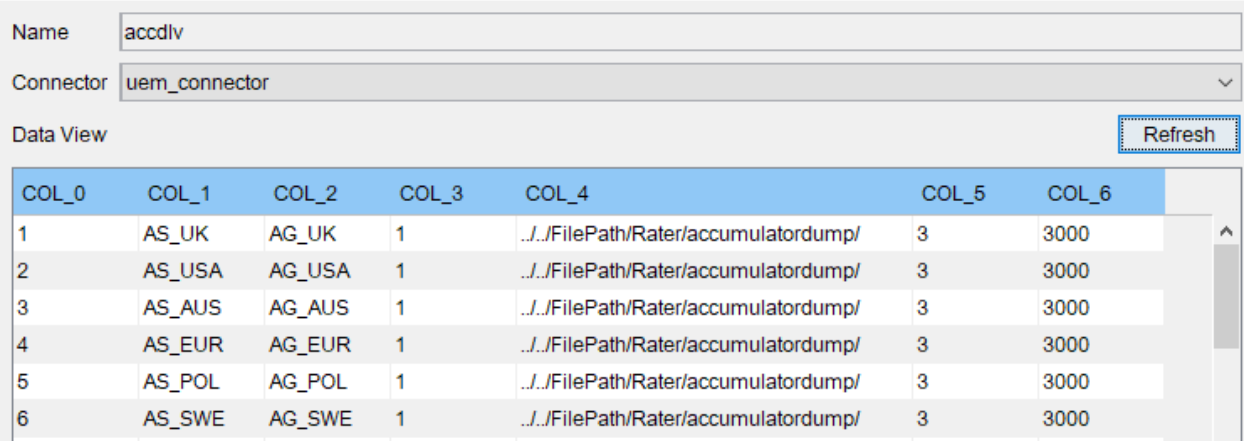


Figure 154 - Custom DataSource editor

Name	Name of the Data Source
Data View	Dynamic render of the DataSource
Connector	The Class Connector node that is going to provide the Class template and object instance.

3.11.13 Memory Data Source

A Memory Data Source loads up data of an underlying data-source in memory – for faster processing.

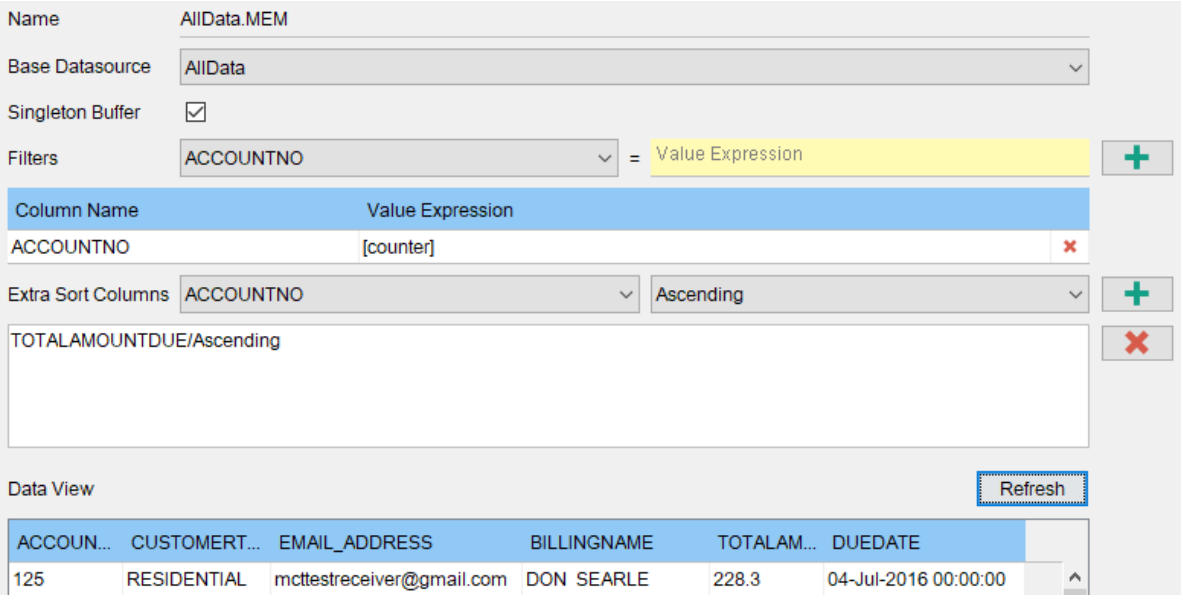


Figure 155 - Memory Datasource editor

A memory data-source will load all data from an underlying DS in a shared buffer (it can be per-thread or globally shared). Once loaded, access is super-fast and additional filtration can be applied using binary traversal.

Notes:

- Memory Datasource can be used to replace SQL sub-queries (queries having 'where clauses' with a field from another query). Since each sub-query generates a network hit, a SQL Query without the 'where clauses' and a Memory Datasource built on top of it with the relevant 'Filtration Criteria' (which is equivalent to the 'where clauses') will perform much faster under all circumstances.
- Memory Datasource consumes memory – and hence, users may need to configure the –'Xmx' JVM parameters as required. It depends on the size of the data being loaded.
- Filtered DS over a Memory DS vs Filtered Memory DS: For fast processing of complex filtration criteria, it may be required to configure a Filtered DS over a Memory DS. It will be fast, but not as fast as the Filtration that happens inside the Memory DS. Filtration inside the Memory DS works by sorting the data set and performing a Binary Search which is exponentially faster than the linear search performed by Filtered DS.

Name	Name of the Data Source
Base Datasource	Name of the Datasource whose data will be loaded in memory
Filters	Filtration Criteria: • Name of the DS column • Value expression that should appear on that DS column When the Memory DS is accessed (using any one the Data Source Fields linked to this DS), the Filtration will be performed already.
Extra sort columns	The filtered data set can be further sorted by specifying the soft columns and sort order. More than one sort column can be specified and sorting is done in the same sequence as they are defined.
Data View	Dynamic render of the DataSource

3.11.14 Transformer Data Source

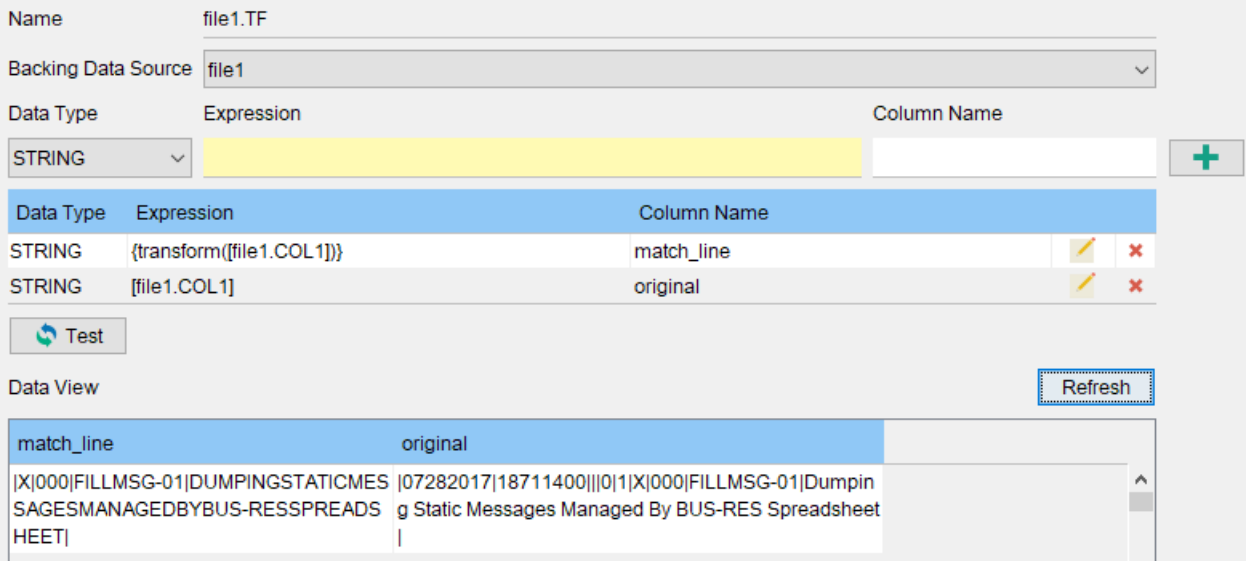


Figure 156 - Transformer Data source editor

A Transformer datasource creates a new Data Source based on another existing dadasource. It allows using PARE Expressions to define the value of the user defined columns. While it was always possible to transform Data Source Fields, the Transformer datasource allows loading up the transformations in memory and apply in-memory filtration.

Name	Name of the Data Source
Base Datasource	Name of the Datasource whose data will be transformed on-demand

Data Type, Expression, Column name	Each column of a Transformer datasource is defined using: • PARE DATATYPE – type of the data in the column • Expression – PARE expression that defines the value. One or more Datasource Fields from the Base datasource can be included in the Expression. For numeric columns, arithmetic operators can be used, and for Boolean columns, conditional operators like <, ==, etc. can be used. • Column name – Name of the column bearing this Datatype and Expression. One or more such user defined columns can be configured.
Test	Test if the Datasource can be rendered – i.e. the Base datasource is working and the Transformer column names are available.
Data View	Dynamic render of the DataSource.

3.12 Models

PARE Models support creation of user defined Objects containing one or more configured Fields or other Models. By defining Models, all the included data can be retrieved at once as an object (for local use), or a JSON (not remote transfers).

The simplest way to think of models is as Java Objects. For each Model defined within PARE, user can see the corresponding Java source code that is dynamically generated. PARE however does not directly provide Java source files to users. Instead, those files are compiled and packaged as a JAR. All Models within a Configuration are stored in a JAR file and the user application can use those to include the models in user apps.

3.12.1 Model

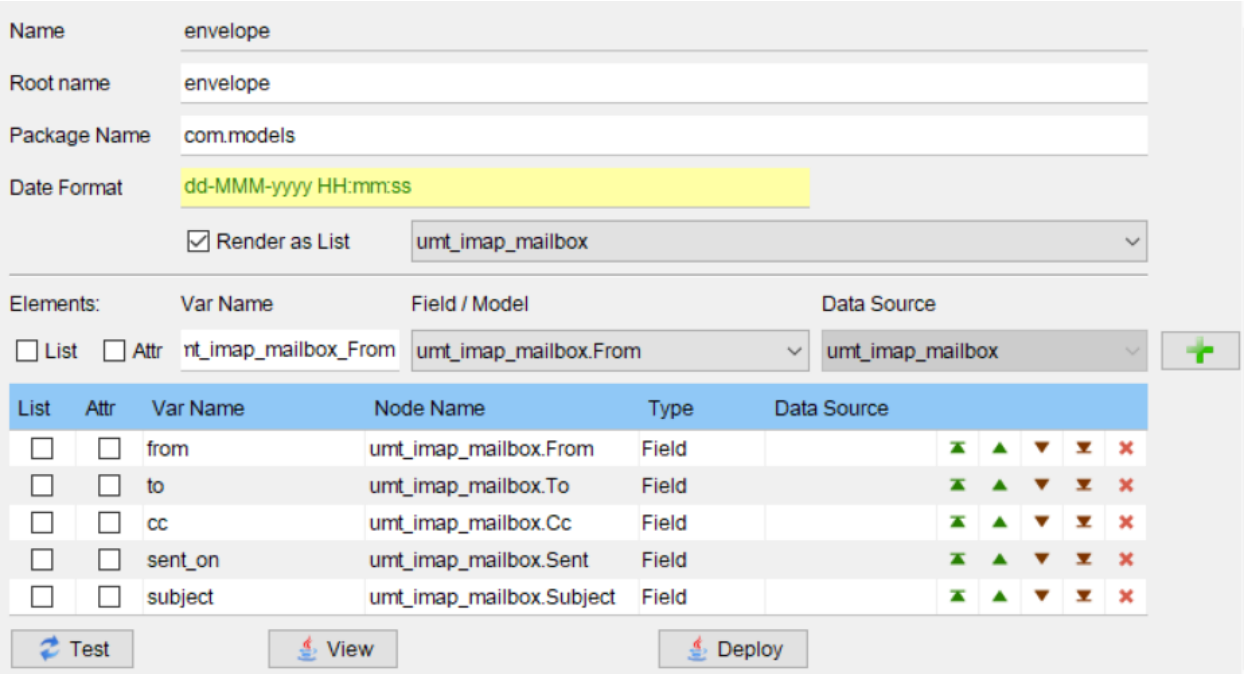


Figure 157 - Model editor

Models represent data structures. A model can include either Fields or other Models. User can choose to create a List or a Single Model member. List members needs to be backed by a corresponding Data Source so that all the List items can be calculated. Typically for Data Source Fields, in List mode, the backing Data Source is auto-selected. For other Fields and Models, user selected Data Source can be applied. If a data source field is included in Single

mode, the value returned in the Model will be based on the current Cursor position in the data source.

Name	Name of the Model
Package Name	Name of the Java Package for the Model. **Note**: This has the following impact: 1 – corresponding 'package' statement in inserted in the Java source file 2 – corresponding directory structure is auto-created inside the Jar
Date Format	Format of Date fields in the XML or JSON serialization
Render as List	Checkbox indicating if a List type element is to be included. If true, a backing Data source needs to be selected for the entire Model.
Element	Consists of the following: • Var name: Name of the variable to be used during serialization • Field name / Model name: Select the field or model to include as element in this model • Data source: For List type element, backing Data source is required to render the list of values • Is List: Sets the list flag. If true, backing Data source needs to be set. This is auto-set for DSF fields.
Test	Extract JSON corresponding to the Model object. **Note**: This will work correctly, only once 'deploy' is done successfully after any changes to the Model.
View	Show the Java source corresponding to the Model.
Deploy	Create / update the Models JAR file with the latest changes in the current Model.

Models support following assets:

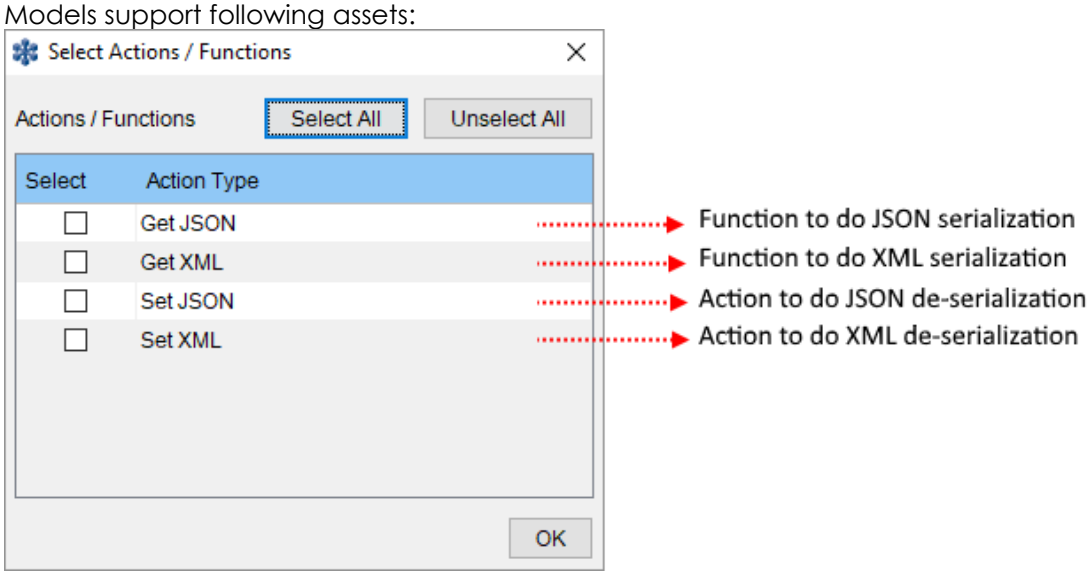


Figure 158 - Model Assets

3.12.2 JAR content

The Models JAR contains all the Class definitions created in a single PARE configuration. All Model elements are converted into 'public' member fields in the class. The classes and fields are named automatically using the following convention:

Name conversion: All ' '(space) chars are replaced by '_'(underscrore). All '.'(dot) chars are replaced by '_'(underscrore).

Model Class Name: Converted Model node name
Member field type: As per PARE type (STRING to String, DATE to Date, TIME to Date, …)

Member field var name: Converted Field node name (can be overridden)
Member Standard Model Type: Converted Model node name (can be overridden)
Member List Model Type: List <packagename+Converted Model node name>
Member Model var name: v_< Converted Model node name > (can be overridden)

Since Models now allow setting of 'var' names, users can easily override the auto-generated names so that it matches with the external JSON and XMLs.

3.13 Plugins

Plugins enable bundling of functionality. PARE Plugins typically consist of an XML and a JAR. The XML contains configuration of the different Plugin nodes and the JAR contains the supporting code that implements the Plugin functionality.

For example, UMT already provides an EMAIL plugin, which enables sending of emails using a configurable SMTP server.

Once a Plugin is installed, all configured nodes are shown under the plugin node – so that main application node lists are not cluttered up.

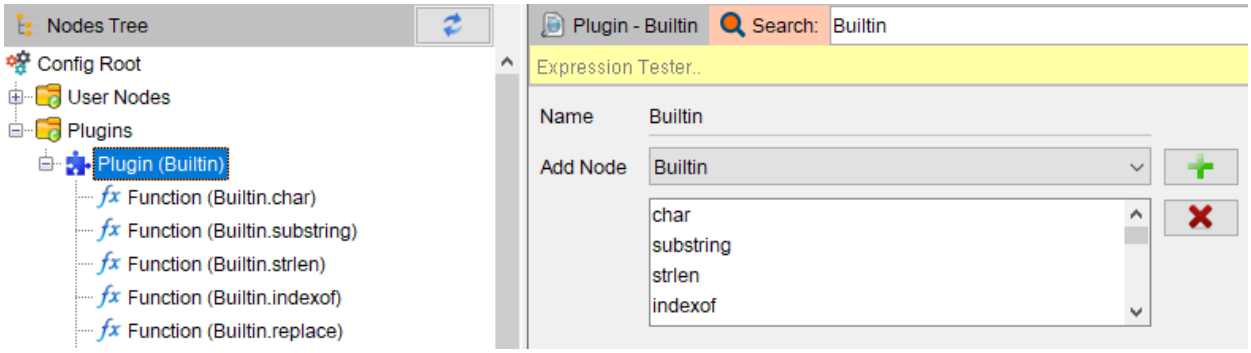


Figure 159 – Plugin View

3.13.1 Plugin Installation

Plugins typically consist of an XML and a JAR file. Note: There may be plugins that are built entirely using built in functionality of PARE, in which case no separate JAR may be required. Once a Plugin is provided, users will need to do the following to include the plugin in their existing configuration:

- Copy the Plugin XML and JAR in a folder
- Open PARE Studio and open the PARE XML where plugin needs to be installed
- Import the Plugin XML. Check if Import is successful – user should see a new plugin node created and the child nodes listed under it. In case a JAR exists, corresponding Class Connector nodes may also exist – please follow the Plugin Documentation.
- Edit the Class Connector and re-map the corresponding Class from the JAR
- That's it – save your configuration and you are done. You can now start using the plugin.

3.14 Linking another PARE configuration

This feature makes it possible to link an existing PARE configuration with another. Users can selectively include the operational assets of another PARE XML in current configuration, thus creating a flexible way to modularize the design.

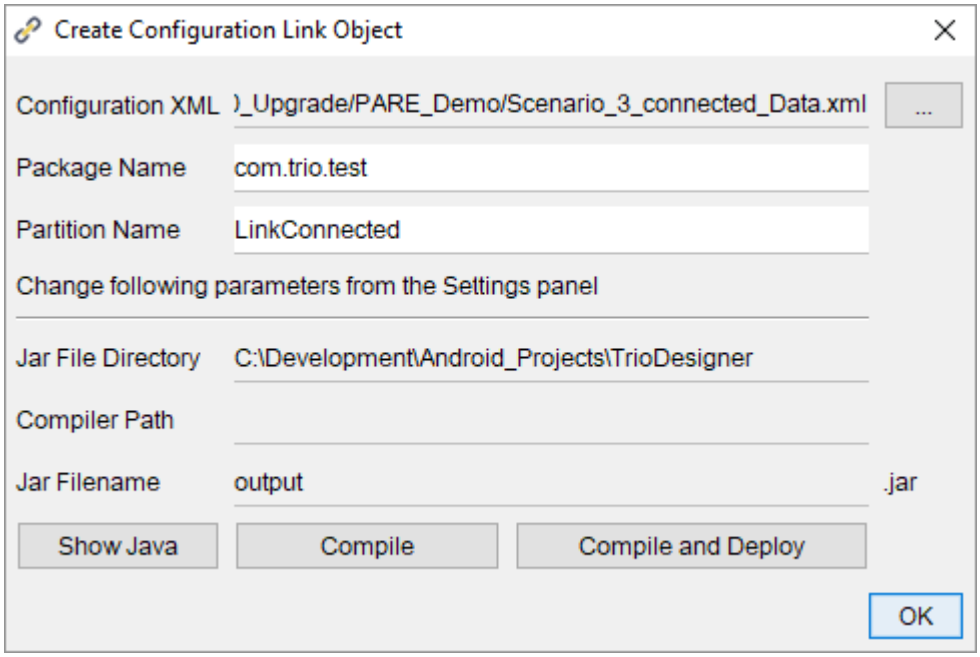


Figure 160 – Configure Link to external configuration

Both configurations (current and linked) execute in their own name-space and interactions happen through a mediator object which is auto-created and loaded as a Class Connector during linking. This Class Connector includes the following resources from the linked PARE XML:

- Fields
- Functions
- Actions

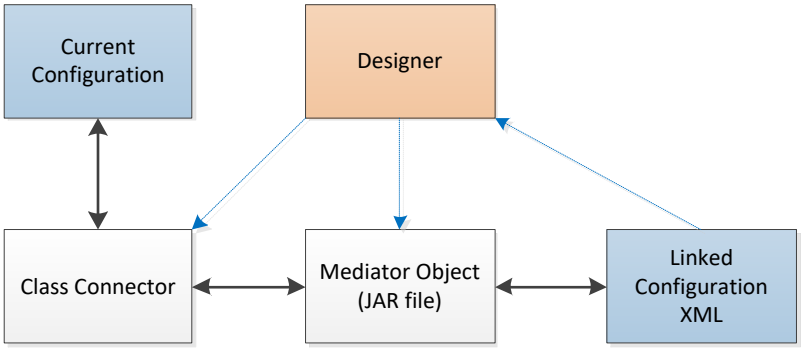


Figure 161 - Creation of Linked Configuration

Configuration XML	File path of the PARE XML to be linked
Package Name	Name of the Java Package that will be used to create the Mediator Object.
Partition Name	The PARE partition name that will be used to load the linked XML. This should be different from the partition name used to finally load the current configuration. Partition name ensures that the current and linked configuration stays in their separate name spaces.
JAR Directory, Compiler Path, JAR filename	Path where the Mediator Object will be created. This is shared with Models JAR path. Editing is possible from the Models List panel (refer section 3.9.1).
Show Java	Shows the auto-generated JAVA file that will be used to create the Mediator Object.
Compile	Compile the auto-generated JAVA File and creates the JAR file.
Compile and deploy	Compile the auto-generated JAVA File and creates the JAR file and

	auto-creates a Class Connector using the JAR. Users should use this button to complete the linking and then use Class Connector facilities to selectively expose assets of the Mediator Object.

3.15 Node List

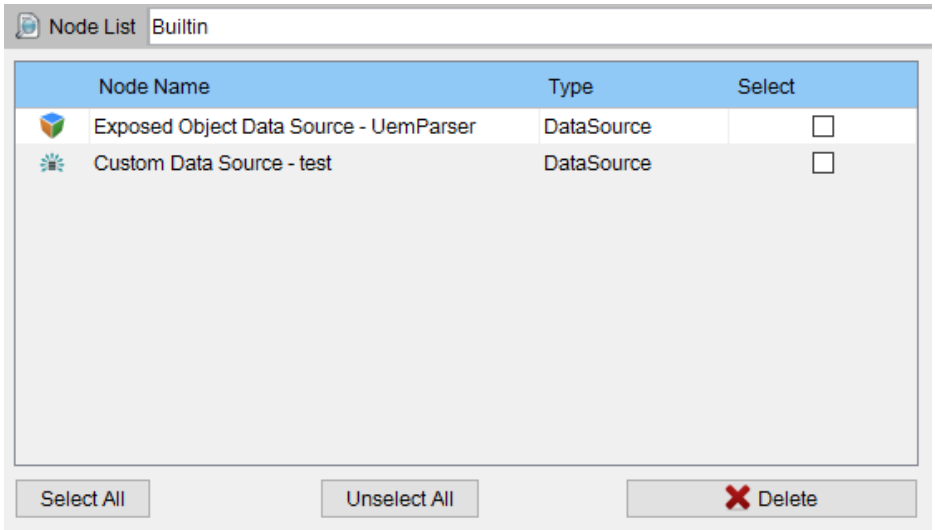


Figure 162 – Node list viewer

Node List panel is displayed whenever any of the Type nodes are clicked. For example, the Nodes with following names: Fields, Actions, Conditions etc. This panel allows user to review and remove all un-used nodes. This is used for cleanup.

Note: Deleting a Node needs to satisfy 'do dependency' condition, i.e. no other Node in the same Engine Instance should be depending on the Node being deleted.

3.16 Node Properties

Node Properties helps attach custom properties to any Node within PARE. This helps in providing application flavors to different Nodes so that they can be treated in application defined ways. For example, a custom property may be used to determine if a particular Field's value is shown in the Application UI, or which particular Action may be fired in response to a button click. In general, it allows loose coupling of Nodes to end Application behavior in fully user defined ways.

Node properties can be set using the Properties editor invoked from the right-click menu of all Nodes:

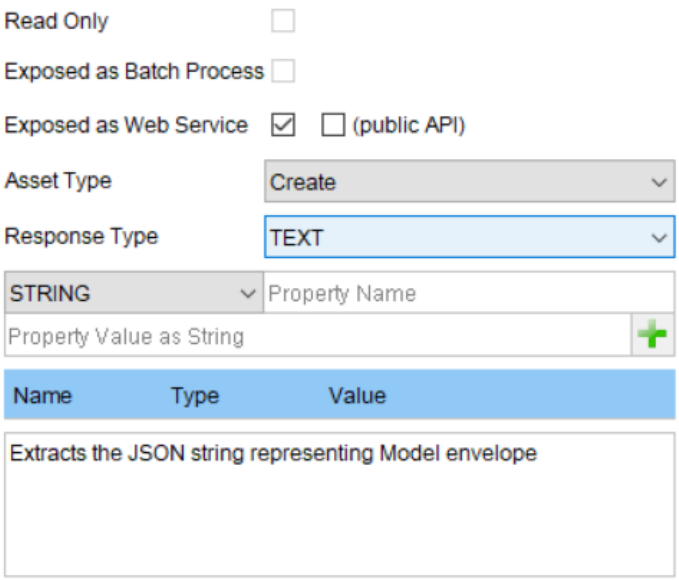


Figure 163 – Properties editor

Read Only	Indicates if the Node's Read-Only state is turned on
Exposed as Public API	Indicates if the selected Asset (Action, Function or Field) is marked as a 'public' API – means, Login token is not needed to access this Asset in PARE Server. Note: REST API Flag must be turned ON for turning this flag as ON.
Exposed as REST API	Indicates if the selected Asset is exposed as REST API once the configuration is deployed on PARE Server.
Asset Type	Indicates the Asset type – Create, Read, Update, Delete. This corresponds to the HTTP method used to access the Asset REST API – POST, GET, PUT and DELETE.
Response Type	The Response Media Type of this Asset: text/plain, text/html, application/xml, application/json
Property Name	Name of the Property
Data Type	PARE Data Type of the property (STRING, INTEGER, …)
Value as String	Value of the Property as String (automatically converted to target data type by PARE)
Properties View	Shows the properties data in tabular format. Also allows deletion of rows.
Micro Service API	Flag to determine availability of an asset (Action / Function / Field) as Micro Service API.

4 Integrating with PARE

There are multiple ways of integrating PARE in your IT landscape. We will discuss the following scenarios here:

- PARE Java API integration
- REST API Integration via PARE REST API
- Existing asset integration via Class Connector
- Existing Web Service integration via PARE SOAP and REST Client

4.1 PARE Java API integration

PARE Java API allows direct integration of the PARE library with your Application. It's the most versatile and powerful integration option (PARE Studio uses the Java API to create the XML configuration). Using the Java API, user has full flexibility to load one of more configurations, operate them or even alter the configurations on the fly (although altering configurations at run-

time is not recommended for general users). The Java API integration is also the fastest as there is no Network layer in between the application and the PARE orchestrated actions.

Here we will discuss some very basic use cases of a user application, but for details of the API are available in Javadoc format.

Note: Almost all Java API methods throw Exceptions – so your code must follow standard exception handling mechanisms as mandated by the Java language.

Create Instance and Load a configuration

First step for accessing PARE is to create an instance and loading up a configuration. Method is as below:

```
RuleManager rm = RuleManager.getInstance("myapp");
rm.load(new FileInputStream("C:\\Users\\User\\Desktop\\myconf.xml"));
```

Directly calling PARE Actions or Functions from Java

Let's assume a sample configuration like below (myconf.xml):

```
<rulesconfig version="1.5.0">
  <action breakonsuccess="false" help="" methodname="runCmd" name="test_action"
type="Sequence">
    <param datatype="STRING" help="param1"/>
    <param datatype="INTEGER" help="param2"/>
    <subscriber action="sysout" params="[param2]: [param1]"/>
  </action>
</rulesconfig>
```

An Action named 'test_action' is defined that accepts two parameters, viz param1 (String) and param2 (int). To invoke the Action from Java code using PARE Java APIs:

```
// Get the Action handle
Action act = rm.getAction("test_action");

// Set params and execute he action
act.setParam("abc", 0); // set the parameter at position 0 (first)
act.setParam(123, 1);   // set the parameter as position 1 (second)

// execute the action
act.execute();
```

Using PARE Expressions

PARE Expressions are one of the most powerful and versatile features and users can take advantage of it directly from Java code.

```
// Call the Expression parser with desired expression string
ExpReplacer.getInstance().replace("{test_action(abc, 123)}", rm);
```

Note:

- Parameters are first converted into Strings and then added to the expression.
- This method adds versatility as Actions, Functions and Fields can be evaluated in same way.
- Expression parser always returns a String – so user needs to convert it back to intended type.

4.2 Using a HTTP Client (OkHttp) to access PARE REST API

Any existing Web Service Client can be reused to connect with PARE assets. The PARE REST API provides 2 ways:

- Using the evaluator
- Using the asset selector

Login to PARE Server (and acquire an access token):

```
OkHttpClient client = new OkHttpClient();

Request request = new Request.Builder()
  .url("http://localhost:8080/RuleServer/services/login")
  .post(null)
  .addHeader("Authorization", Credentials.basic("username", "password"))
  .build();

Response response = client.newCall(request).execute();

// store the token and use in the later calls
String token = response.body().string();
```

Example of calling the asset selector (single node is called):

```
request = new Request.Builder()

.url("http://localhost:8080/RuleServer/services/email/assets/umt_imap_mailbox.Subjec
t")
  .post(null)
  .addHeader("Authorization", "Bearer "+token) // token used here
  .build();

response = client.newCall(request).execute();
System.out.println(response.body().string());
```

Note: Users can use any REST Client (like Postman) to configure the above for testing purposes. PARE itself can be used to connect with a remote PARE server using the Rest Service Function.

4.3 Legacy asset integration via Class Connector

Integrating existing Java assets into PARE flows is easy:

- Create a Class Connector
  - Specify the Java asset location
  - Select the class
  - Select the Constructor parameters
  - Select the instantiation type
- Map the methods as PARE Actions or Functions
- Integrate them in PARE flows

Refer to this section for details.

4.4 Existing Web Service integration via PARE SOAP and HTTP Clients

When using PARE as a service composer, following features will be used to connect and orchestrate with existing API assets:

- Rest Service Functions (as HTTP API client)
- SOAP Functions (as SOAP Client)

5 PARE Server

PARE Server is an advanced API Gateway that allows calling of selected (aka. exposed) PARE Assets (i.e. Fields, Functions or Actions) via HTTP Web Services.

5.1 Overall Architecture

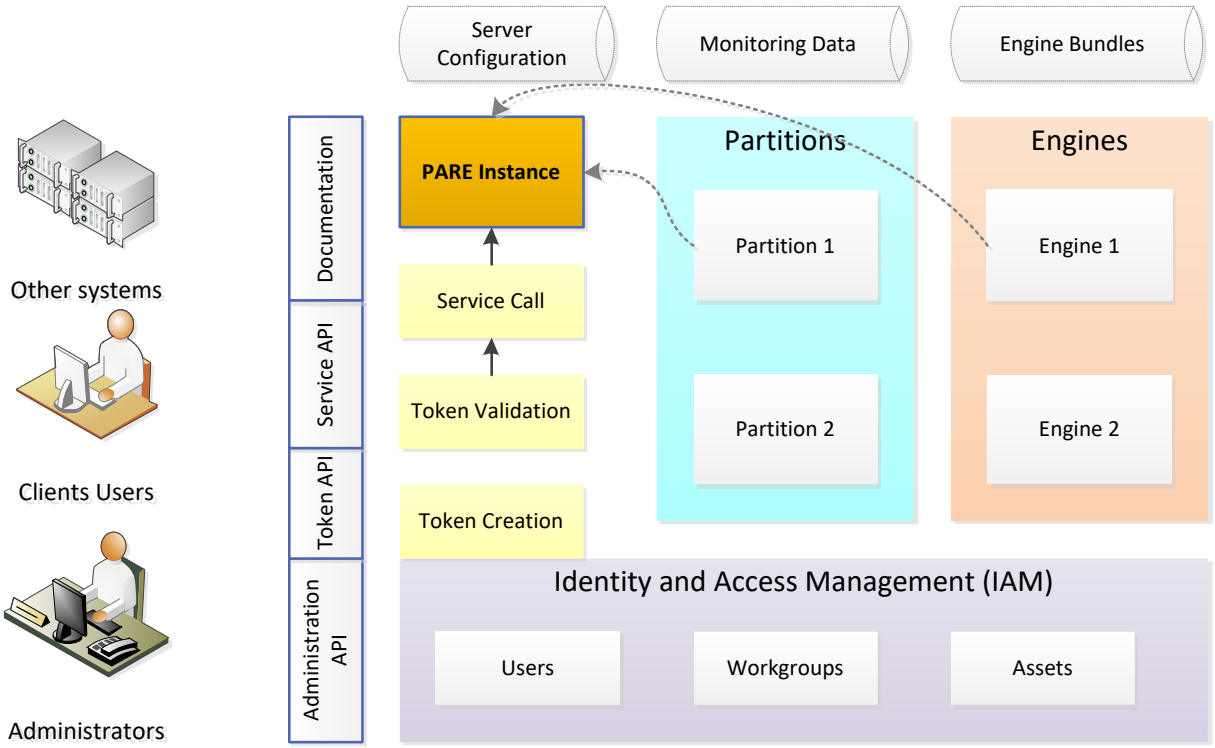


Figure 164 - PARE Server Architecture

5.2 Core concepts

PARE Server acts as a container for one of more PARE configurations and exposes configured Assets from those configurations as HTTP web services. PARE Server also provides IAM functions via configuration of Users, Workgroups and Assets that control which User has access over which web service APIs.

5.2.1 Engines

A PARE configuration that is deployed onto a PARE Server is called an Engine. Multiple PARE configurations (engines) can be deployed in a single PARE Server. Engines are the modular functional blocks and each Engine can support one or more API endpoints as configured in the PARE configuration.

Please refer to here for details about process of marking an Asset as API.

Engines are used to create PARE Instances on-demand by PARE Server and each Engine can be used to create one or more PARE Instances depending on the number active Partitions configured in the Server.

Refer here for details about Engine deployment process.

5.2.2 Partitions

The concept of Partitions remains the same as defined here. PARE Server allows a User to be connected to only ONE partition. If that User makes an API call, a relevant PARE Instance created using the corresponding Engine and Partition is used to serve the call.

Since multiple environments can be simulated within the same PARE server using Partitions and since a User are tied to a single partition, this facility can be used to implement multi-tenant API within PARE Server.

5.2.3 Access Control

PARE Server's IAM (Identity and Access Management) module defines the concept of Users, Workgroups and Assets.

The Access control structure is as follows:

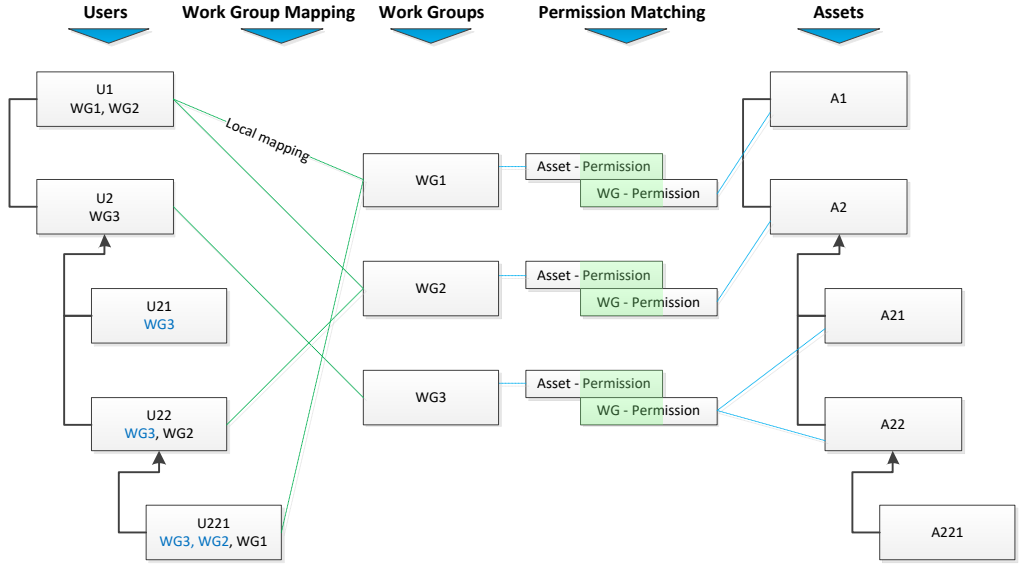


Figure 165 - Access control structure

Note: PARE server does one-way permission resolution – i.e. the Assets (named engines) does not declare any specific permission for any workgroup.

Salient features:

- Assets are equivalent to Engines (mostly, as there are few special Assets which indicate special privileges – like Administration privilege and supervisor privilege.
- Each Workgroup defines permission to one or more Assets (Engines). The permissions per Asset include:
    - C – Create – Create / POST type APIs of the engine can be called
    - R – Read – Read / GET type APIs of the engine can be called
    - U – Update – UPDATE / PUT type APIs of the engine can be called
    - D – Delete – Delete / DELETE type APIs of the engine can be called
    - M – Manage – Manage / POST type APIs of the engine can be called
- Each User can be assigned with one or more Workgroups

Built-in structure:

The 'admin' user having 'administrator' workgroup having CRUDM access over 'controller' asset is the built-in structure that cannot be changed. However, password of the 'admin' user can be and should be changed after PARE Server installation.

If any User is granted the 'administrator' Workgroup, she will have super-user access. Since, more than one super-user is not recommended, other supervisors can be granted permission to the lower level 'engines' asset that allows them to add or remove named engines. The super-user (having all access including creation of new users and workgroups) and the supervisor (having permission to manage deployed engines), forms the admin team per PARE server instance.

5.2.4 Access Tokens

PARE Server uses stateless JWT tokens for access control. This method offers easy scalability and security and is current state-of-the-art for accessing web services. Once the Access Token is granted by the IAM module, the client can make Service calls using the Access Token without

any dependency on the IAM itself. Hence its possible to segregate Authentication and Token generation capability from call processing capability in a multi-node Cluster setup.

The Access Tokens have a lifetime though and needs to be renewed from time to time. It is also possible to 'logout' a user – which invalidates all existing Access Tokens issued to that user this far, and all clients using that user's token needs to be re-login again. The logout feature is typically required when a user's permission structure has changed. Please refer here for the GUI details.

Access Tokens have imprint of all permissions that the User has got and that is used to determine what Service calls the user can make.

5.3 PARE Server HTTP API

Note: Swagger documentation of all API is available from /apidoc/index.html

Error! Hyperlink reference not valid.

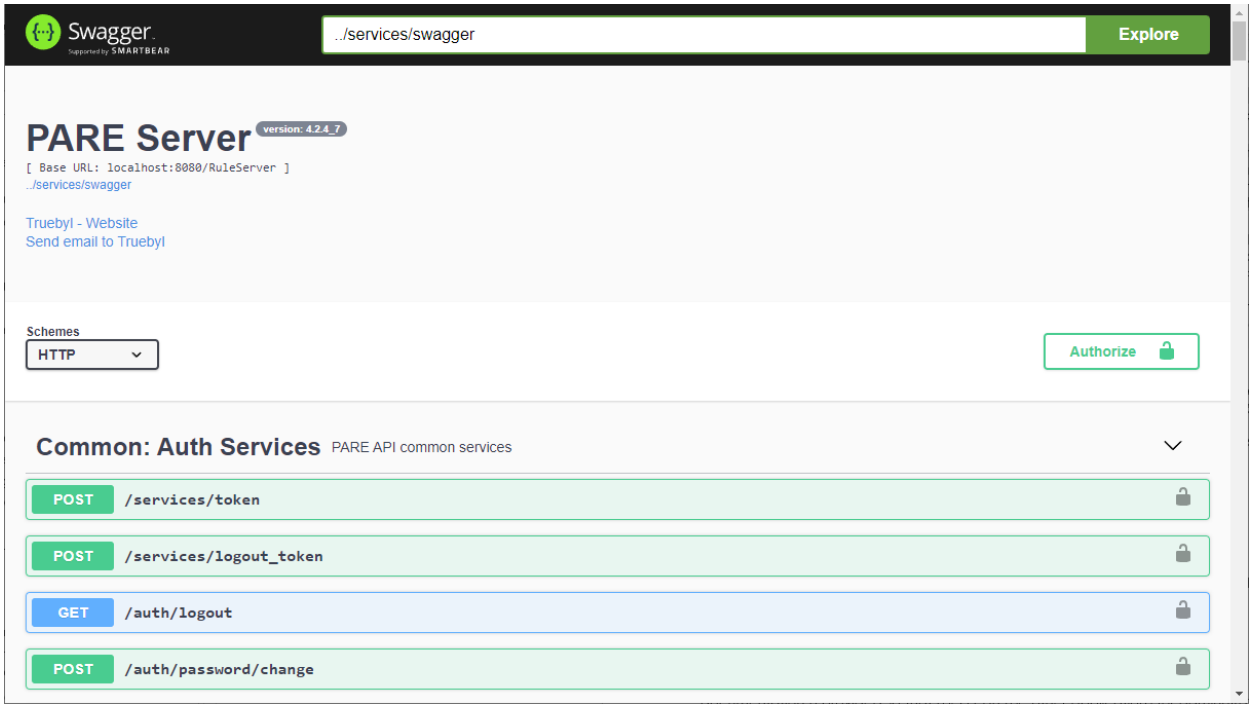


Figure 166 - Swagger Documentation

5.3.1 Hardening PARE Server Security

When deploying PARE in a production environment, one should follow these steps as mentioned:

Note: Typically, this should be done by an implementation engineer in consultation with a Customer's representative. No need to do all this for development setup unless it's meant for testing the security features themselves. When setting new Passwords or Keys, PARE enforces a limit of 10 chars. However, its recommended that at least 20 chars are used for better security. Using a password generator service like https://passwordsgenerator.net/ is recommended.

1. Change the KeyStore password - this is safe and has no impact on existing assets. It makes the KeyStore inaccessible to attackers.

2. Change the Password encryption key (note: this must be different from the KeyStore password) - note that once this is done, unnamed passwords (that are directly encoded in XML) in existing configurations may become unusable. Using named passwords is the

recommended approach as it takes care of environmental differences and different encryption keys can be set for different environments.

3. Change the user encryption key (note: this MUST be different from the password encryption key) - but make sure that the same key has been set for all environments. In general, using Encrypted User Fields is NOT recommended, as it does not offer much in terms of security. If Encrypted User Fields are not used, this step can be skipped.

4. If Pare Server is being used, configure the Servlet Container (commonly Tomcat) to use a SSL Certificate (CA signed is recommended, but self-signed is also OK for testing) and HTTPS Only configuration (by turning off or redirecting the HTTP Connector).

5. Do not share contents of the PARESettings folder with anybody or perform copy of folders across environments. Bulk transfers of Host Parameters and Named Passwords across environments are now possible via Server Links. Individual modification of a remote Host Parameter or Named Password is also possible via Server Admin GUI.

6. Create different passwords and keys for each environment. Exclusion: Refer to point 3.

7. Keep a secure backup of the currently set keys/passwords per environment. This enables changing the keys on a regular basis.

8. Initially all key/passwords are set as "default" - change it!

9. If the KeyStore becomes damaged, the same can be deleted from PARESettings folder. However, PARE GUI will have to be restarted and steps 1 to 4 performed again, and in addition, one will need to re-enter the value of all passwords in PARE Password Manager.

6 Optimizing for performance

- **When to use Memory DS**: If there are sub-queries (a DB query that uses a Field in where clause from another Query) that are frequently used on a stable (data does not change much) main-query, Memory DS is a good option to drastically improve sub-Query performance. In this case having the Sub-Query configured without the relevant where clauses and doing an in-memory filtration is much faster. However, if the Memory DS requires frequent reloads (as the main query output itself is very dynamic), you may re-consider your design. It may also make sense to configure a Transformer DS between the DB Query and Memory DS, so that only relevant information in right format gets loaded into Memory.

- **When to use Filtered DS**: Filtered DS is powerful but not fast. If in a scenario, the size (number of filtered rows) of the Filtered DS is quite large and batch processing on it is needed, one may take a different approach that avoids Filtered DS altogether: Using a Rule, with same condition as the Filter, on the base DS and taking actions only in the TRUE block. This approach is much faster. Filtered DS on smaller data sets has lesser impact on performance.

- **When to use MT Jobs**: Multi-threaded jobs can allow up to 100 threads per job. However, users should take following precautions:
  - Do not create a Thread tree, i.e. call a MT job within another MT Job, or call a MT Job within a Loop. That will create a large number of threads that will slow down processing and increase thread creation->switching->destruction overheads. Threading should be used only once in a process, and at or near the top level, so that thread creation is done only once when the process is invoked.

  - Size the number of threads properly - an Action taking more I/O time is more suitable for multi-threading, than say an Action that's taking more CPU time. PARE makes it

easy to change the number of threads and monitor total execution time and CPU/Memory utilization. Increasing the number of threads will not proportionately shorten execution time, and users should select optimum number threads that gives optimum benefit in terms of time and resources (leaving > 25% CPU for others is a thumb rule).

- **DataSource performance optimization**: Most of PARE DataSources has a 'Volatile' flag. If this flag is checked (the default), the DS will check if underlying data needs a refresh (refresh-check) at every access. This can be a performance drag in some cases (if the PARE Expression that defines data content, like a Query or file path, takes longer to resolve). If the 'Volatile' flag is unchecked, refresh-check at every access is skipped, which can improve performance. However, it also means that user is now responsible for informing the DS when data refresh is really required. This is typically done using the 'DS.Refresh' Action.

- **Using 'Native Evaluator'**: If there are large number of Condition Expression evaluations in a configuration (Condition Expressions can exist in Condition nodes, Rules, Decision Tables or general expressions), checking 'Native Evaluator' flag turns on the PARE embedded evaluation parser. Otherwise, the evaluation is delegated to the embedded JavaScript engine. The Native Parser has restrictions like it does not allow Math expressions directly (use 'Compute' built-in) but is faster than the Script based evaluator.

7 PARE Workflows (BPM Engine)

PARE now comes with an integrated BPM engine for coordinating both automatic and human tasks.

7.1 Concepts

PARE Workflows are fundamentally a collection of Tasks. These Tasks are executed as per the designed flow, either in sequence or parallel or both, when an instance of the Workflow is started. Once all the Tasks reach a terminated state – which can be either "finished" or "cancelled", the Workflow itself is marked as "finished" as well.

7.1.1 Task States

Presently there are four (4) task states:

- TO START – default state when a task is initialized.
- STARTED – when a Task is started.
- FINISHED – when is Task is completed.
- CANCELLED – when a task is cancelled (considered as finished without being executed).

When a Workflow Instance is created, the initial state of all Tasks is "TO START". Each Task is then evaluated for its readiness to be executed. Tasks will no triggers are started immediately in parallel. Subsequently other tasks are started as their trigger conditions become valid.

Please note that for Manual Tasks, the engine attempts to "assign" it to eligible users and the state remains "TO START". Its only after someone "accepts" a Manual Task, that its marked as "STARTED".

7.1.2 Relationships (Triggers)

Each Task can also be related to other Tasks. For example, Task 2 may have a F-S (Finish-to-Start) relationship with Task 1. In this case, Task 1, which does not depend on anything, will start when the Workflow starts, and Task 2 will automatically start when Task 1 is finished. The Workflow will finish when both Task 1 and Task 2 are finished (or cancelled).

These relationships give PARE Workflows its flexibility and ability to model actual processes. Each such relationship is called a "Task Trigger". Following types of triggers are supported (note: a task can have zero triggers also – which is default situation when a Task is first created):

- Finish to Start (F-S) – Child task is triggered when parent task is finished.
- Start to Start (S-S) – Child task is triggered when parent task is started.

Any given task can have multiple F-S and S-S relationship with other Tasks within the same Workflow. Note that the system checks and prevents circular relationships.

7.1.3 Task Types

Workflows are meant to coordinate human tasks. If there's no human task in a business process, probably PARE Orchestration features are enough to configure them. PARE Workflows supports both Human and Automatic tasks – since most business processes need both.

Following Task types are supported:

- Manual – Tasks that require human inputs. Examples of this type of tasks include installing a CPE and noting down the serial number, taking a meter reading, performing an inspection, or submitting a report or delivering a package. Manual Tasks are usually associated with skillsets so help allocate the right person for the job. Each Manual Task has a duration defined in either Hours, Minutes or Seconds.

- Automatic – Tasks that do not require any human input. Examples of this type of task include executing business rules, performing a database update, calling an API to trigger another system, processing a file or image, acquire data from another system.

- Waiter – Tasks that wait for a certain external event (ex: wait for an email to arrive) or wait for fixed amount of time, or a combination of both. If both fixed time and event conditions are configured, you also need to correctly configure "whichever earlier" or "whichever later" flag.

- Flow – This is a wrapper Task around another Workflow so that it can be called as a Task too. This helps in modularizing the flows and eases building of complex flows by joining other workflows. The effective duration of a Flow Task is equal to calculated duration of all the Manual Tasks in *Critical Path* of that flow.

Below is a visualization of a sample Workflow with the different Task types and Relationships shown (note: BPMN notations are used):

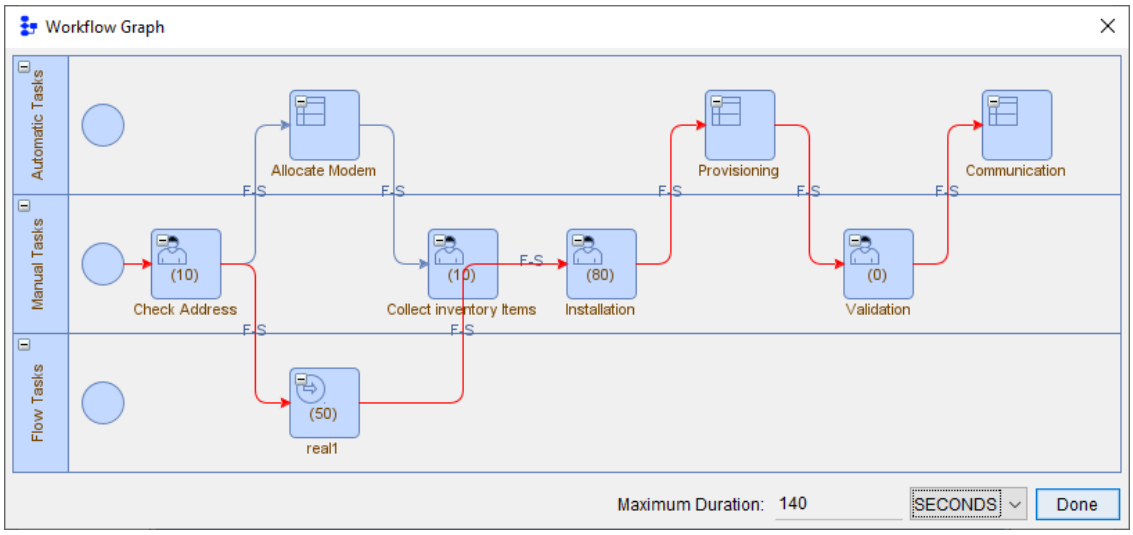


Figure 167 - Workflow Visualization (the RED line indicates Critical Path)

7.1.4 Workflow Data

A business process makes sense only when it is associated with the right data. PARE Workflows offers powerful and flexible features to set and get data securely during the Workflow lifecycle.

Data Template definition

At the Workflow definition level, the data fields are defined:

- Field name – Name of the Data Field.
- Content Type – Can be Text, Computed, Choice, File, Barcode.
- Datatype – All PARE data types are supported for "Text" and "Computed" type fields. For others, the data type is always STRING.
- File Type – For "File" content type, following file types are supported – Text, Image, Custom.
- Choice Items – For "Choice" content type (should show a drop-down in UI), choice items can be configured (comma separated strings).
- Launch Param – Indicates that value of this Field is provided when the Workflow instance is created. Note that a Computed field cannot be a Launch Param.
- Editable – Indicates whether the Field is editable or a constant. Note that a Computed field cannot be Editable.

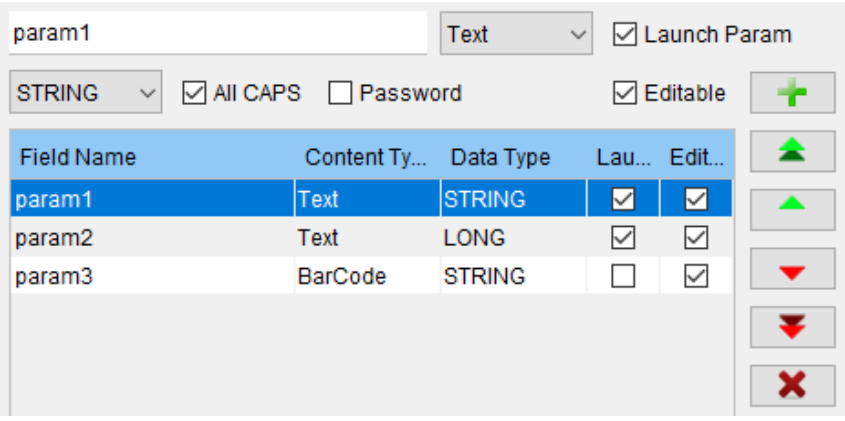


Figure 168 - Workflow Data Template

Data Access

All Automatic Tasks have access to all data elements. They can set or get any data element during their execution. Built-in Functions and Actions are provided to read and update instance data from within the Automatic Task orchestration:

- getWorkflowData (key) – returns the value of a data element.
- setWorkflowData (key, value) – sets new value for a data element.

These built-ins only work within an Auto Task executing in a Workflow instance context.

Manual Tasks, on the other hand, have "edit" access to only certain fields – which are set as "Editable Fields" during Manual Task template configuration.

Note that a Workflow Data Field can be updated only if its marked as "Editable" in the template configuration.

Flow Task data mapping

Since Flow tasks are other Workflows which has its own data template, the definition of a Flow Task includes the field mapping between the parent Workflow and the child Workflow. This is achieved by associating one field name of the parent with another of the child. An unlimited number of such associations can be made per Flow task.

Note: The data association between a child and parent data works both ways. All associated data is first fed into the child task via Launch Params. Thereafter, any change in the Child

instance data fields are automatically synched back into the parent through the mapping relationship.

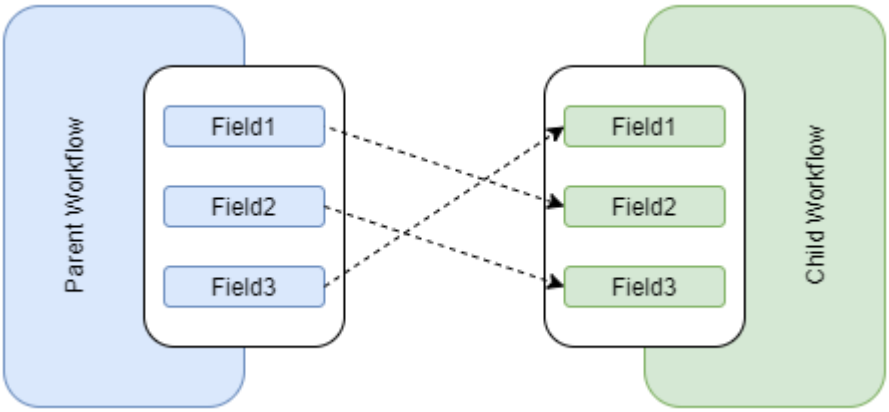


Figure 169 - Inter workflow data transfer mapping

Data persistence and lifecycle

Data values are associated with a Workflow instance at the time of its creation. After that, all changes to the data elements are persisted until the Instance is "finished" and subsequently removed – on expiry of the retention period - from the live tracking data repository. All changes to Instance level data are synchronized, immediately persisted, and made accessible to all tasks of the workflow during its lifecycle.

7.1.5 Conditional Branching

This facility allows an "automatic task" to select alternative execution paths within the workflow. This is done by "cancelling" one or more downstream paths. So, it's possible now to implement scenarios like:

a) Cancel further processing of the workflow after a point - i.e., Cancel all downstream paths

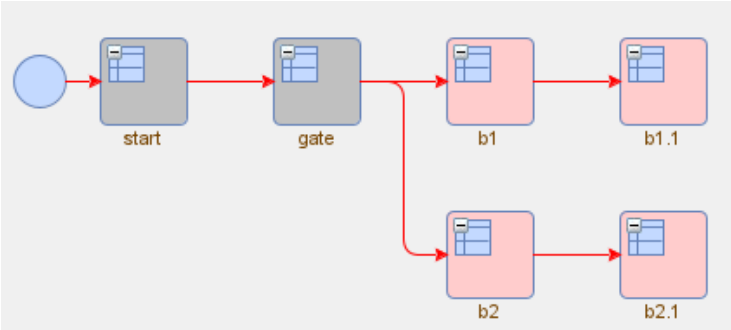


Figure 170 - All paths cancelled

b) Cancel only a subset of the downstream paths - simplest case: just cancelling one of two alternative paths

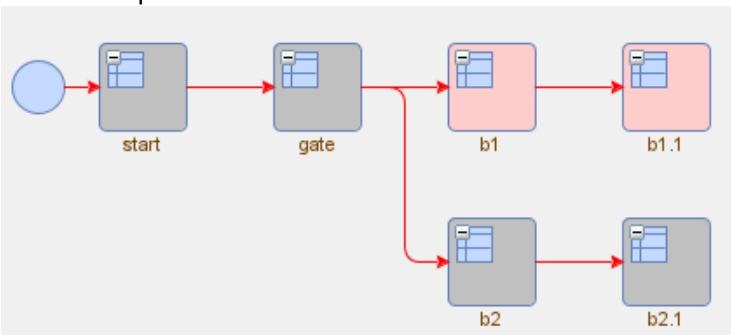


Figure 171 - Alternative Path chosen

c) Cancel an interim branch within downstream paths

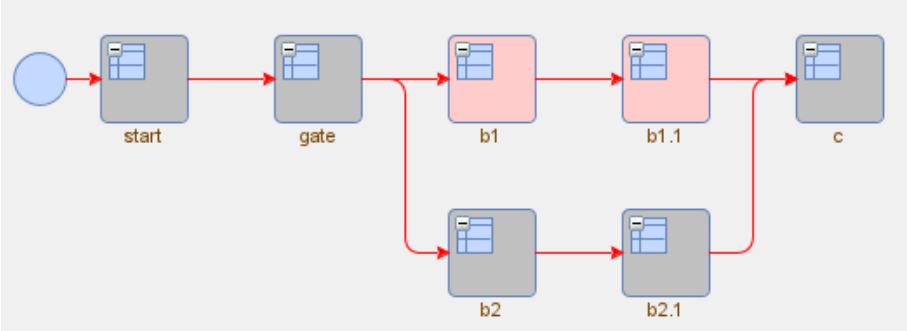


Figure 172 - Interim branch cancelled

Cancellation of a path is done in a data driven way using a new built-in {cancelWorkflowTask(taskname)}, which can be configured in any Automatic Task. This gives full flexibility to implement decision making rules at the template level. Only the starting task of a Path needs to be cancelled using the built-in – rest is taken care of by the BPM engine.

Any type of task can be cancelled - Auto, Waiter, Manual or Flow.

7.2 Workflow Editor

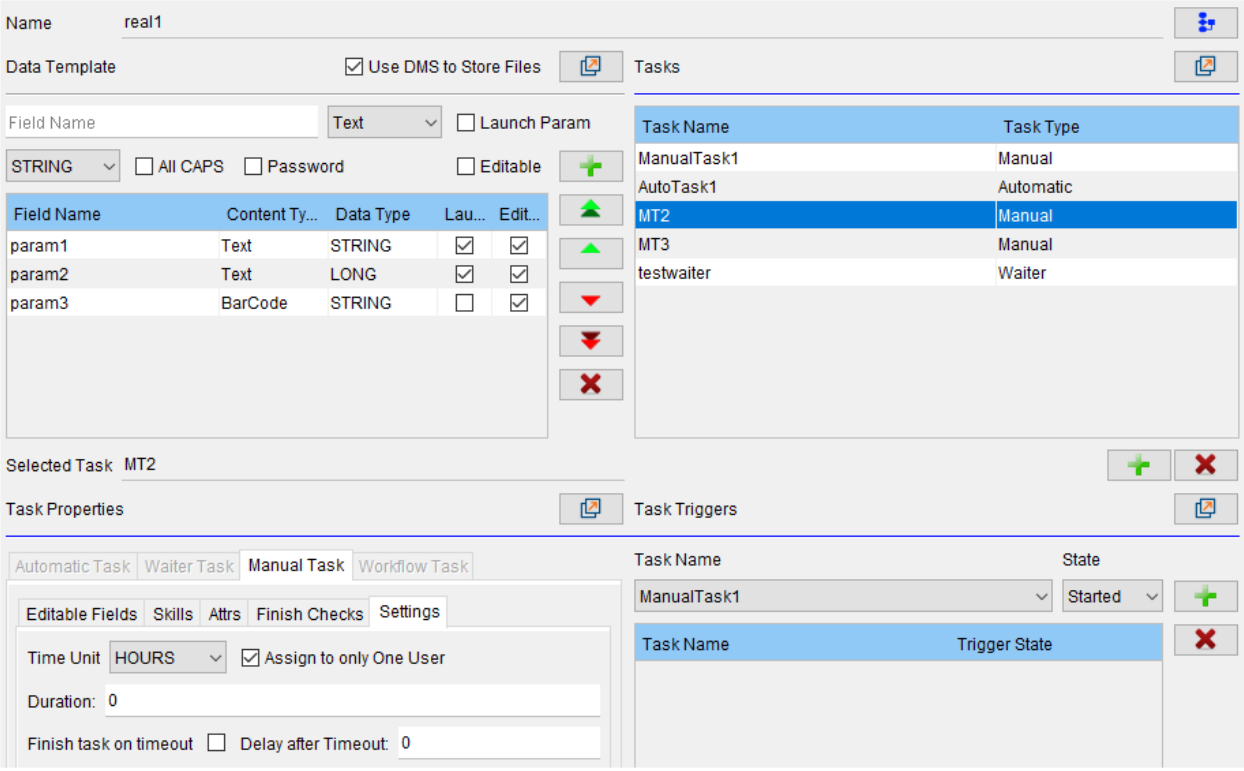


Figure 173 - Workflow editor

The Workflow editor allows creation of a Workflow Template including all the contained Tasks, Task Triggers, Data template, Data Mapping for flow tasks and Task content (auto task actions, Manual Task skills and editable fields and similar.

7.2.1 Manual Task

A key objective of the workflow engine is to help coordinate and execute Manual Tasks as part of a Business Process flow.

PARE BPM supports following Manual Task features:

- Editable Fields

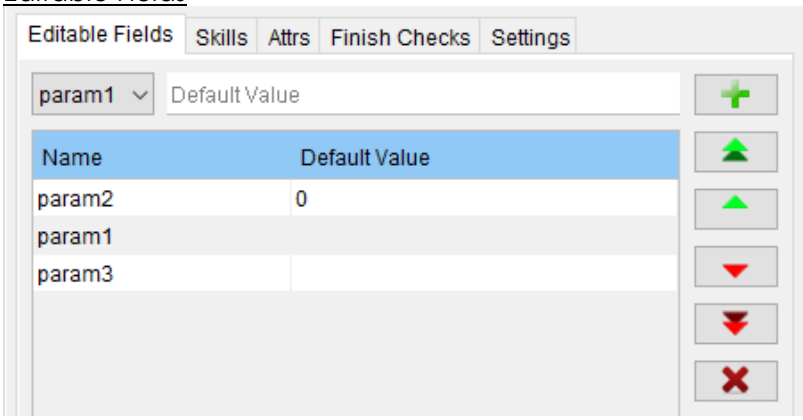


Figure 174 - Manual Task Editable Fields

it's possible to configure in the template which fields the Manual Task needs to fill up while finishing it. For example, if the Manual Task involves installation of a Modem, its serial number needs to be entered into the system when the job is completed. All Editable data fields must be part of the Workflow Data Template. A default value can be configured for all these fields which are automatically set in case of a timeout.

- Attributes

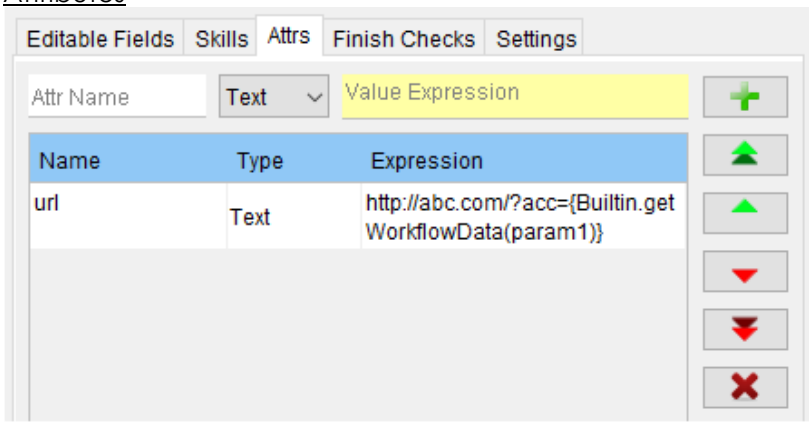


Figure 175 - Manual Task Attributes

These are read-only referencing information that can be configured as part of the template and may be shown by the GUI to the user executing the Manual Task. Example includes reference information like Job address, Contact numbers, allotted vehicle number and similar such info.

- Skills

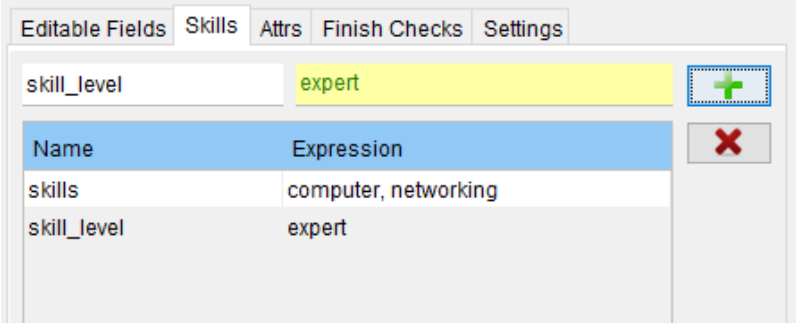


Figure 176 - Manual Task Skills

One or more key/value pairs indicating the skills required to perform this Manual Task. For example: "skill/network-engineer". When a Manual Task becomes eligible for execution, the Workflow Engine searches users with matching skills (in user Attribute settings) and assigns the Task automatically.

- Finish Checks

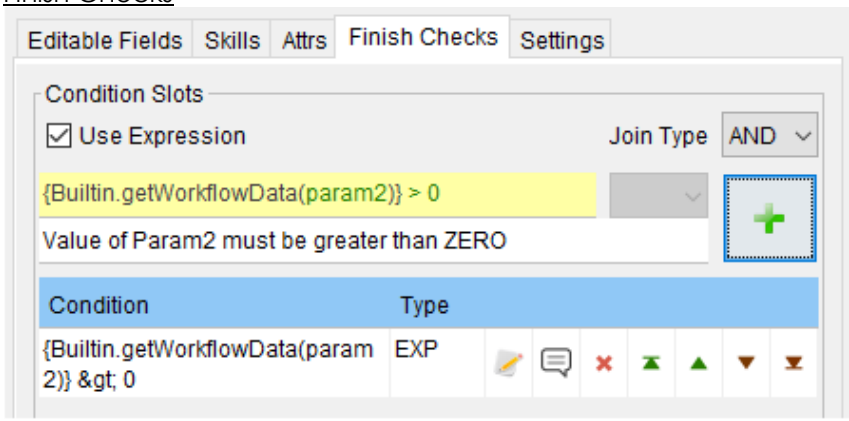


Figure 177 - Manual Task Finish Checks

Conditions to be checked by the engine to ascertain that a Manual Task is indeed been completed properly. These conditions can enforce that Modem serial number is entered properly before marking the modem installation task as finished.

- Other Settings:

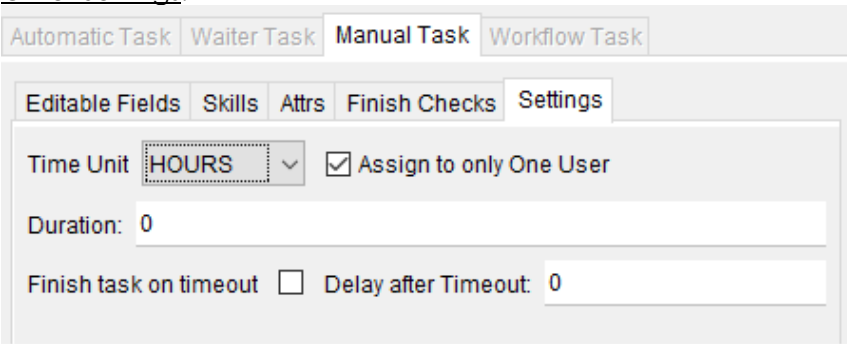


Figure 178 - Manual Task Settings

  - Duration – estimated time taken to perform this task
  - Finish on timeout – whether Task needs to be marked as finished after a timeout period. Note that timeout counter starts once a Manual Task has "STARTED", i.e. someone has "accepted" the task.
  - Timeout Delay – If above setting is "true", time to wait after "duration" is passed before marking the Task as finished
  - Assign to only one user – by default the Manual Task is assigned to all eligible users as per skill match criteria. This flag tells the engine to assign to only one user with appropriate skills.

7.2.2 Automatic Task

Automatic Tasks bring the power of PARE Orchestration within the BPM engine. These tasks are typically used for the following purposes:

- Execute Business Rules and take decisions within the workflow
- Perform computations and update application data from within the flow
- Perform conditional branching
- Dynamically invoke other workflow instances
- Send alerts or notifications as required by the business process

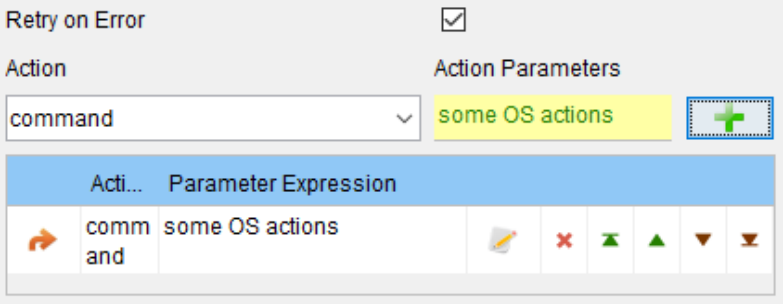


Figure 179 - Automatic Task configuration panel

Automatic Task contains an Action Container – just like a PARE Sequence Action – and can be used to implement your logic. The Action Container comes with standard features like Action Selector, Parameter Expression, and call sequence editor.

Automatic Tasks also come with a "Retry on Error" switch, which tells the engine to keep retrying the sequence of actions repeatedly in case of any error.

If an automatic Task encounters errors, it will not proceed. Its however possible to mark an Automatic Task as "finished" via administrative API.

7.2.3 Waiter Task

A waiter task waits.

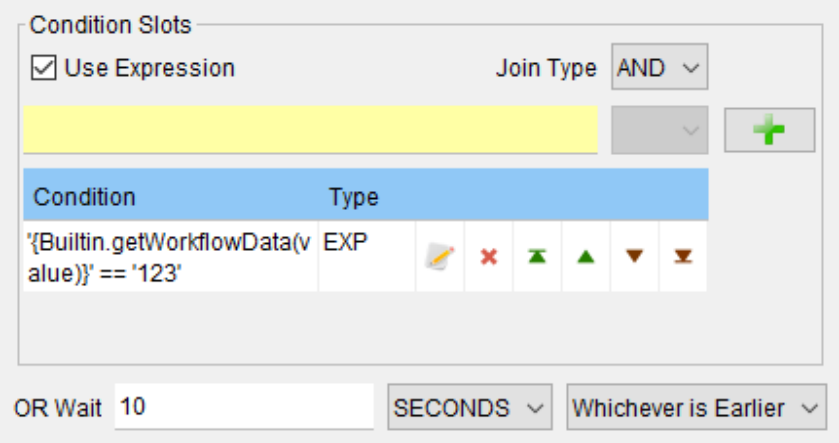


Figure 180 - Waiter Task configuration panel

- You can either wait for a fixed amount of time or you can wait for an external event or both.
- If both fixed time and event conditions are configured, you also need to correctly configure "whichever earlier" or "whichever later" flag

Waiter Tasks are required in following scenarios:

- Waiting for an external event to occur – say an email has been sent, and the workflow must wait until a response is received
- Waiting for a fixed amount of time – say the field tech must travel for one hour and till then, next actions need to be on hold

Waiter Tasks show up with a special icon in the graph dialog:

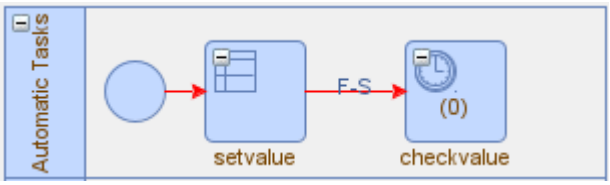


Figure 181 - Waiter Task visualization

7.2.4 Flow Task

Flow Tasks indicates an embedded child workflow. The BPM engine allows embedding of one workflow within another at template level, using Flow Tasks. This helps in modularizing the flows and eases building of complex flows by joining other workflows. The effective duration of a Flow Task is equal to calculated duration of all the Manual Tasks in *Critical Path* of that flow.

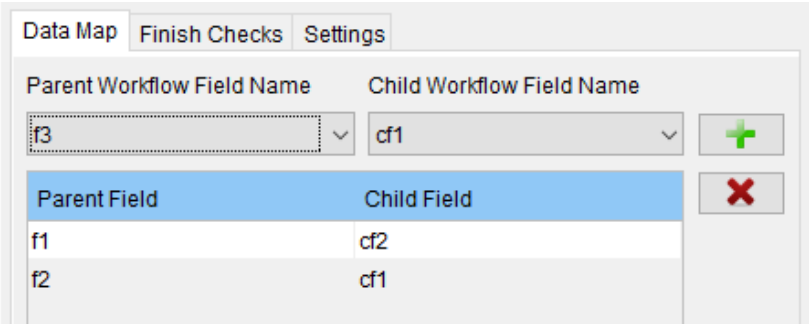


Figure 182 - Flow Task Data Mapping

The Data Mapper specifies which Parent Field is mapped to which Child Field. When a Flow Task is started, the Child workflow instance is launched with the mapped fields used as Launch Parameters. Thereafter, any change done in the Child fields' value are immediately synchronized back to the Parent. Referring to above diagram, if value of Child Field "cf1" is changed, Parent field "f2" is also updated.

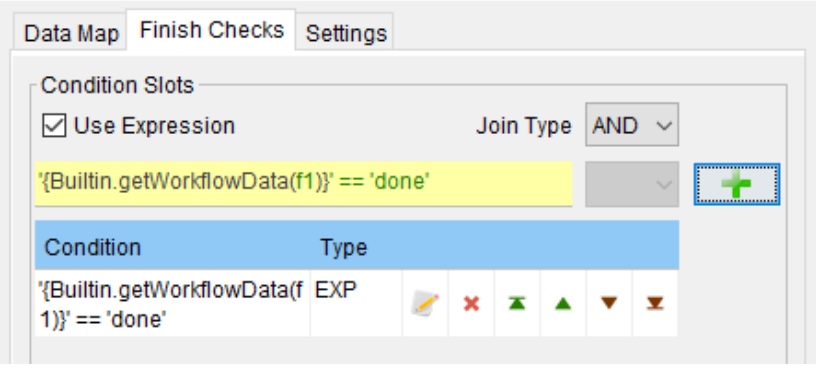


Figure 183 - Flow Task Finish Checks

Flow Task Finish Checks helps specify conditions to check once a Child Workflow has finished execution. By default, if Finish Checks are configured and it returns "false", the Flow Task is launched again.

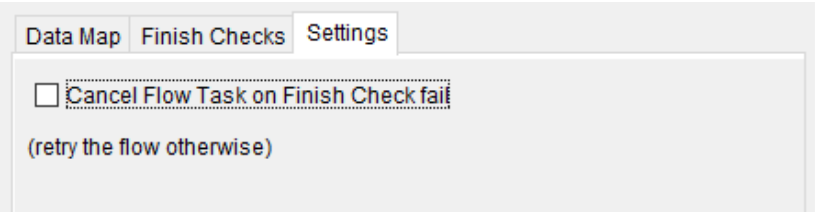


Figure 184 - Flow Task Settings

Flow Task Settings can help change the default Finish Checks behaviour. Template designer can opt to "CANCEL" the Task (and the immediate branch) if Finish Checks are configured and it returns "false".

Flow Task configuration contains the data mapper configuration – refer to section 7.1.4 above for details.

7.2.5 Workflow Visualizer

The visualizer button at the top opens the Graph Visualization window, which offers a user friendly BPMN compliant view of the Workflow configured so far. It also shows the critical path and max duration (along the critical path). Please refer Figure 167 - Workflow Visualization (the RED line indicates Critical Path) for a sample screen shot.

7.3 Workflow Monitor

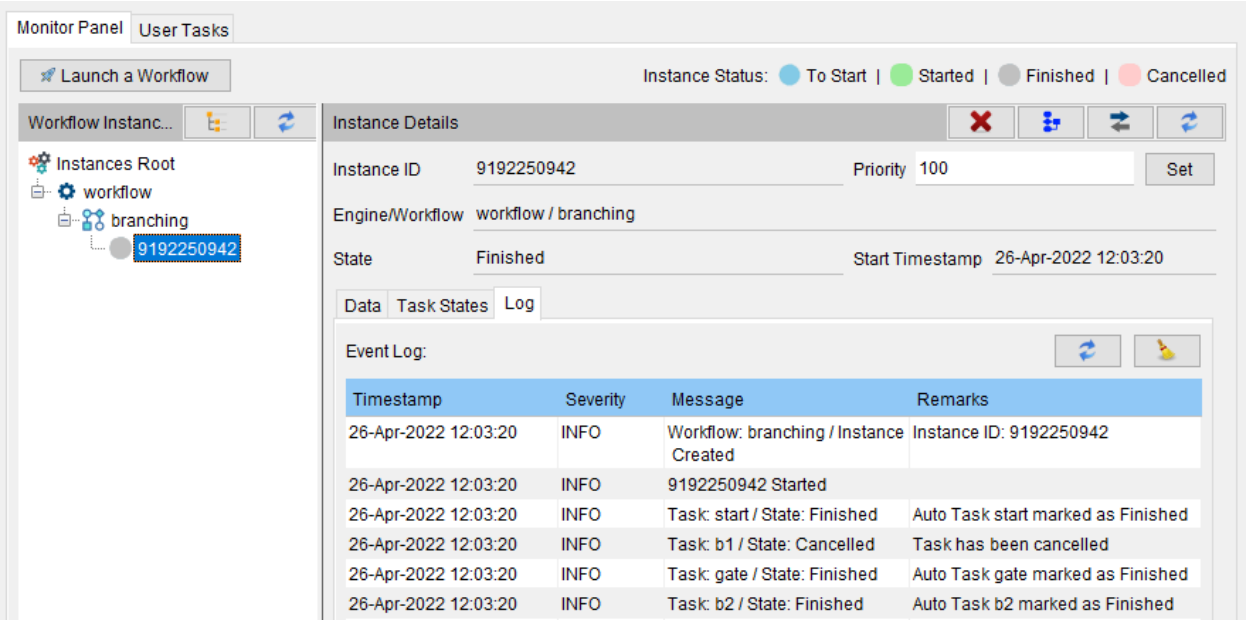


Figure 185 - Workflow Monitor UI Module

PARE Server internally runs a Workflow Monitor process that monitors status of each active workflow instance and progresses them. PARE Studio includes a built-in Workflow Monitor UI Module that lets admins monitor in-flight workflow instances and do troubleshooting as may be necessary.

Note: Only one instance of the PARE Workflow monitor should be running in each environment. The Workflow Monitor thread is not started by the server on its own, but it is started automatically if any Workflow API call is made. Therefore, in a cluster environment, the load balancer should be configured to redirect all workflow API calls to a single selected node.

The workflow Monitor screen has following components:

1) The left navigation area – here a Tree View of all active Workflow Instances per deployed engine is displayed. Each Workflow Instance is represented by its unique ID (UUID) and clicking on any one of them shows the detail view on the right pane of the screen.

2) Detail View top toolbar – The tool bar buttons are used for various common operations, including:
    a. Remove the Workflow Instance – this force removes an in-flight workflow instance.

b. View progress graph view – this button shows progress of the Workflow in a popup window. It's similar to the Workflow template Visualizer, but with progress information and facility to do operations like finishing a task or changing workflow data on the fly.

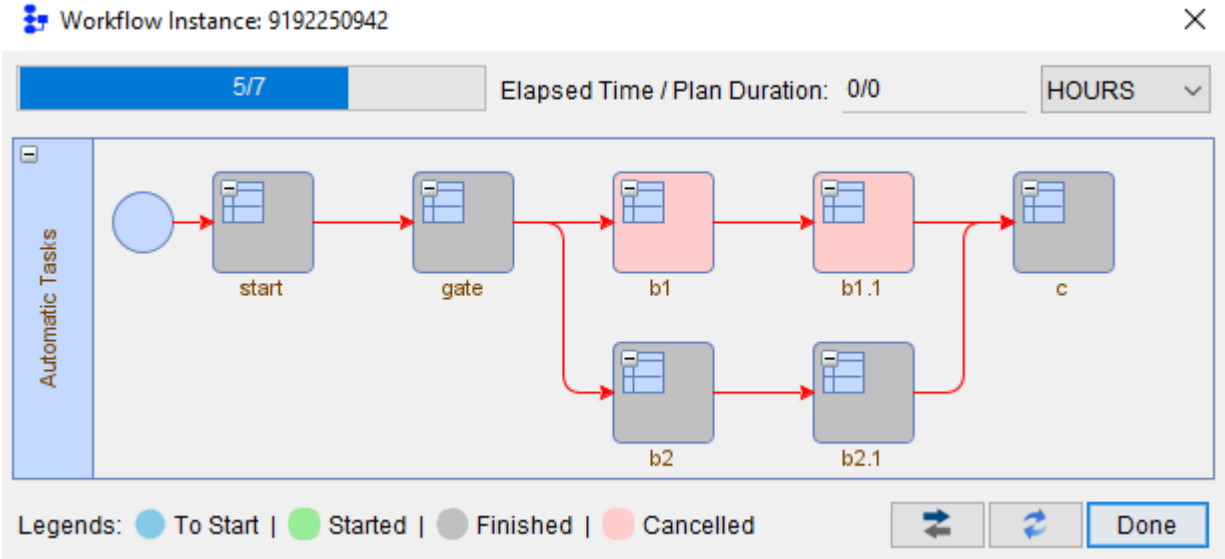


Figure 186 - Workflow Instance Visualizer

c. Instance Data Editor – This button opens the Data Editor screen which allows changing of Instance data.

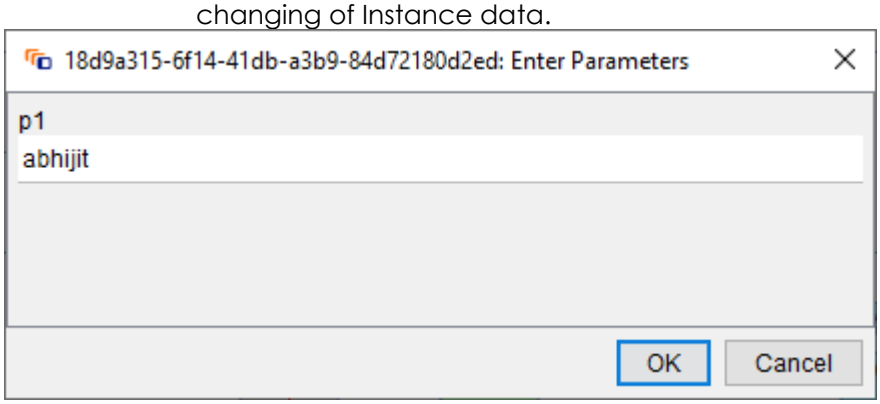


Figure 187 – Workflow Instance data editor

3) Detail view – contains the Data View (actual data values associated with the Instance) and Task Status viewer.

Note: It's possible to refresh status of an Instance – which is a more light-weight operation using the "refresh" button in the toolbar. It's also possible to "refresh" the tree which effectively refreshes information of all the in-flight instances.

7.3.1 Workflow Instance data retention period

Once a Workflow Instance is completed (i.e. "FINISHED"), it becomes a candidate for removal from the live tracking tables. But they are removed once the retention period is over. The retention period is calculated in DAYS after the Instance is marked as FINISHED.

The minimum (also default) RETENTION PERIOD is 3 days.
The maximum RETENTION PERIOD is 30 days.

Users can override the default RETENTION PERIOD by declaring a HOST PARAMETER "`WORKFLOW_RETENTION_DAYS`" and setting it to value between the minimum and maximum RETENTION PERIOD values.

Note that for permanent retention or workflow instance, users should configure webhooks on the "monitor" built-in engine and subscribe to the "workflow" event source. The received payload may be then stored long term in separate storage.

7.4 Workflow Built-ins

startWorkflowMonitor	Starts the workflow process engine for the current partition. Note that the process engine should be started on only one node within a cluster.
stopWorkflowMonitor	Stops an already running workflow process engine.
cancelWorkflowTask	Cancels a child task in the current workflow instance. This built-in is used within an automatic task to implement conditional branching. Parameters: P1 – task name
removeWorkflowInstance	Attempts to remove a running workflow instance. Parameters: P1 – Instance ID
setWorkflowData	Sets value of an editable data field within the current workflow instance. Parameters: P1 – field name P2 – Field value as string
setWorkflowInstanceData	Sets value of an editable data field within the specified workflow instance. This is typically used to do data transfers between parent and child workflow instances. Parameters: P1 – Instance ID P2 – field name P3 – Field value as string
getWorkflowData	Returns the value of a data field in the current workflow instance. Parameters: P1 – Field name
getWorkflowInstanceId	Returns the current workflow instance ID. Parameters: None
getWorkflowInstanceState	Returns the state (either STARTED or FINISHED) of a workflow instance. Parameters: P1 – Instance ID

7.5 Workflow API

All workflow APIs start with /workflows URL path (/RuleServer/workflows/):

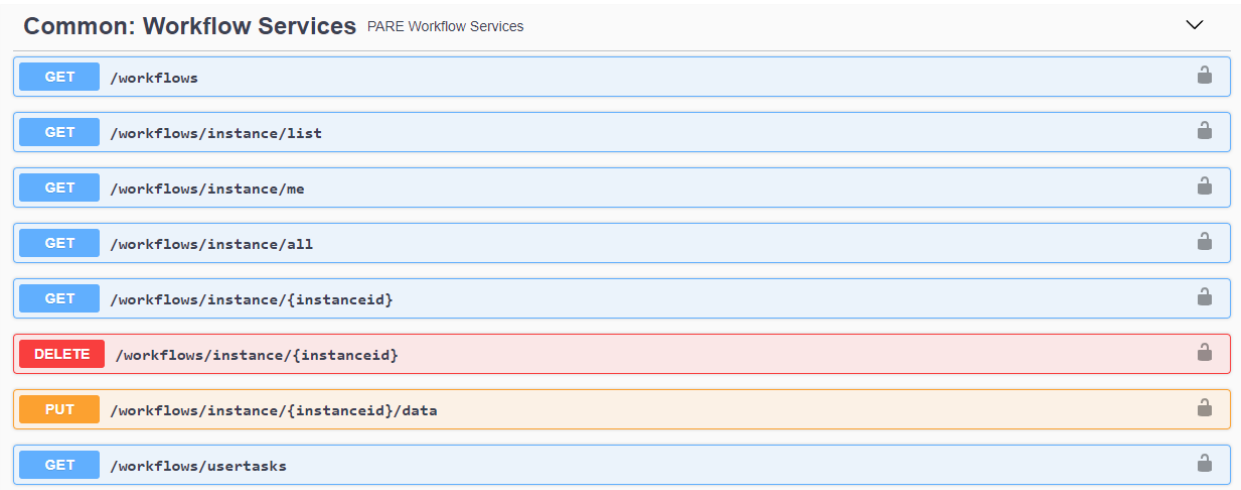


Figure 188 - Workflow Common API Swagger documentation

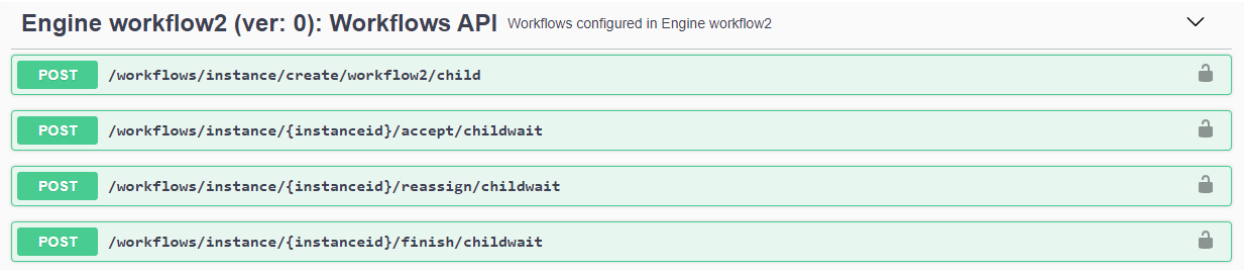


Figure 189 - Workflow Template Specific API

8 Document Management System (PARE DMS)

PARE DMS offers facilities to store, access and search files within a PARE environment. Each DMS entry contains the following information:

- File UID – a unique identifier of the File that can be used to retrieve or access the DMS entry.
- File name – File name including extension.
- File attributes – a set of key/value pairs that sets various application specific file attributes.
- Public Flag – whether the File is publicly accessible or not.
- File content – file content in compressed format.

8.1 DMS feature highlights

8.1.1 Attribute based search

One of the most used features of DMS is the search utility (API) that lets you search all files within a partition for certain attribute values. This is useful to retrieve all File UIDs linked to, for example, a particular account. The search API returns a JSON output as below:


```
[
  {
    "attributes": "{\"default_logo\":\"true\"}",
    "createdby": "admin",
    "engine": "monitor",
    "filename": "ss_logo.png",
    "fileuid": "e168b2e5-d446-49d8-aead-8696b0bf804a",
    "partition": "default",
    "public": "true",
    "url":   "http://localhost:8080/RuleServer/files/default/monitor/e168b2e5-
d446-49d8-aead-8696b0bf804a/stream"
  },
  {
    "attributes": "{}",
    "createdby": "admin",
    "engine": "monitor",
    "filename": "sample.js",
    "fileuid": "7b8af53b-df7f-4a2d-bf5d-a7adb697edec",
    "partition": "default",
    "public": "true",
    "url":   "http://localhost:8080/RuleServer/files/default/monitor/7b8af53b-
df7f-4a2d-bf5d-a7adb697edec/stream"
  }
]
```


The JSON output above includes all relevant file info including an URL to directly download the file.

8.1.2 Access Control

All files are linked to a partition/engine and standard PARE access control rules apply:

- Users of one partition can't access files from another partition unless the file is marked as public.
- Users must have permission on the engine to access files belonging to that engine:
  - Create – To upload new file.
  - Read – To download or retrieve the file content or related information.
  - Update – Update file attributes.
  - Delete – Delete / remove the file.

8.1.3 Common or Engine level storage

DMS can be configured to store all files in a common location. Engines can over-ride this location by declaring their own storage locations. Therefore, it's possible to configure separate storage locations for few selected engines – which may be handling agreement documents for example – while all other engines share the default common location.

Users must make sure that all DMS storage locations are accessible from all Server nodes in a cluster deployment.

8.1.4 Automatic detection of Content Type

The file download URL automatically sets the file content type based on file extension. This helps in proper browser rendering of the file. For example, a JPEG file content type will be automatically set to "image/jpeg" with "content-disposition=inline" so that the image is shown in the browser window instead of offering it to download.

If the file has no extension or has an extension for which any matching content-type can't be found, the default content-type will be used: application/octet-stream, which will prompt the browser to start the download process and may prompt the user with a save dialog.

8.1.5 DMS API

Files API is documented using Swagger and is available in: **Error! Hyperlink reference not valid.**

Search for "File Services".

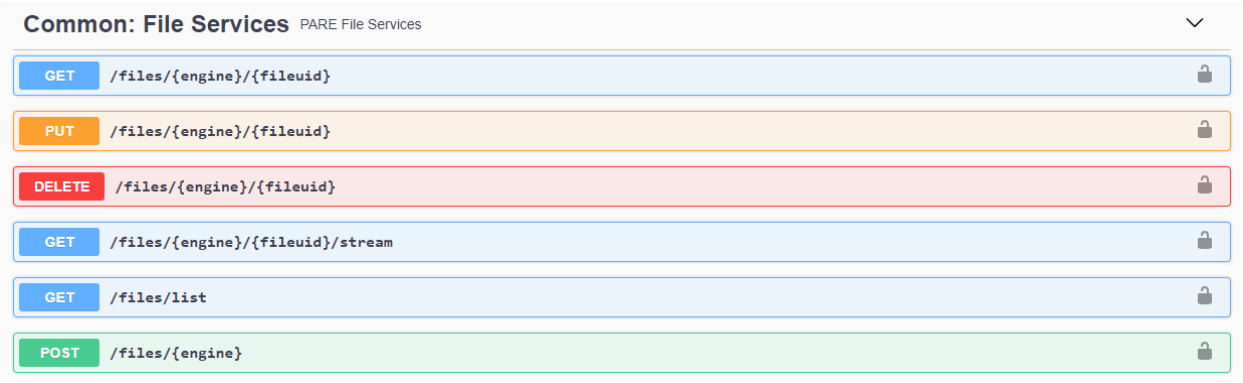


Figure 190 - Files API Swagger Documentation

8.1.6 DMS Built-ins

fileDelete	Deletes a file from the DMS. Parameters: P1 – fileuid
fileSetAttribute	Sets a file attribute. The attribute is removed if the value is set as NULL. Parameters: P1 – fileuid

	P2 – name P3 – value
fileSetPublicFlag	Sets a file as public or protected. Parameters: P1 – fileuid P2 – public (true / false)
fileGet	Gets a FileData object JSON. Parameters: P1 – fileuid
fileInfo	Gets a files listing entry JSON. Parameters: P1 – fileuid
fileUpload	Uploads a new file into DMS. Parameters: P1 – filename P2 – content P3 – public (true / false)
fileDataUpload	Uploads a new file into DMS. Parameters: P1 – FileData JSON P2 – public (true / false)

8.1.7 Studio GUI

PARE Studio provides reference screens that shows features of the DMS.

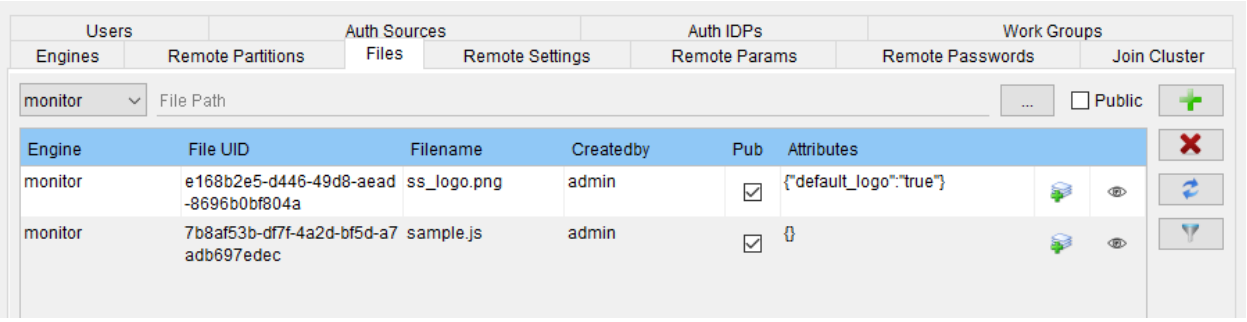


Figure 191 - Files list view

Server Admin -> Files shows list of all files in the partition and all the engines the user has READ access to. It shows the containing engine, the file UID, uploader, public flag and attributes. It also allows downloading of the file – which open the browser with the download URL.

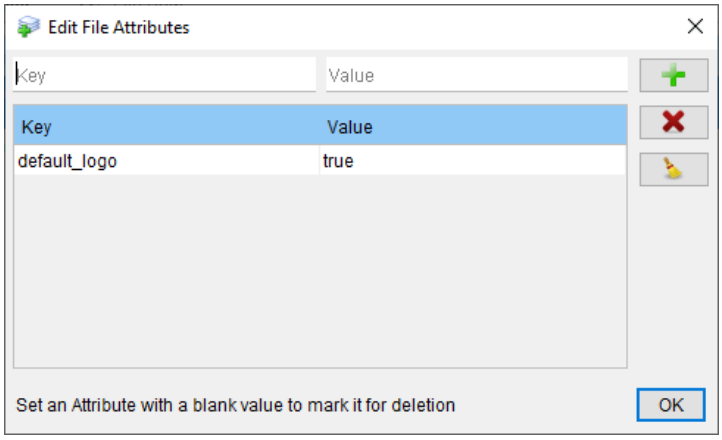


Figure 192 - Set File Attributes

You can set File Attributes through this popup for each file.

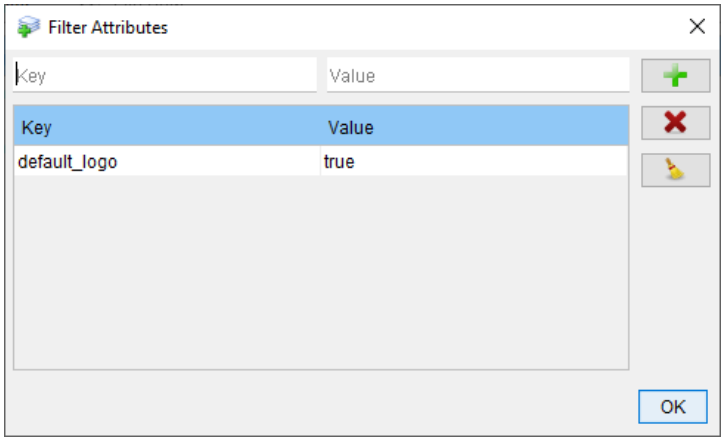


Figure 193 - List Filter

This popup allows filtration of the list using attribute matching.

9 Licensing

PARE offers Host based time-bound or perpetual licensing models. By default, PARE starts in Evaluation mode (which is fully functional) and works for a period of 1 month from the date of first run. Once the license expires, PARE loses ability to SAVE and LOAD configuration XML files. License status can be viewed and changed from the About->Licensing dialog in the top menu bar.

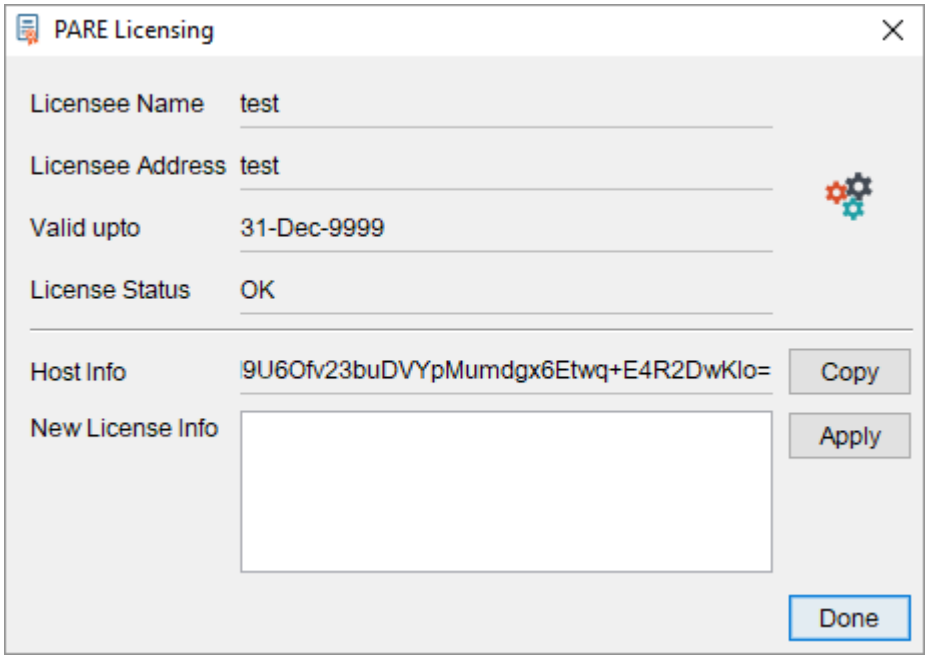


Figure 194 - License Management

For change or renewal of license, user is requested to copy the Host Info text and send it to Usha Martin Technologies, wherein, a valid license key will be provided. The same key needs to be copied and pasted in the New License Info area, and on clicking 'Apply', new License will be installed. In case of any error, existing License will be retained.