Copyright © 1999-2015 Perforce Software.
All rights reserved.

Perforce software and documentation is available from http://www.perforce.com/. You can download and use Perforce programs, but you can not sell or redistribute them. You can download, print, copy, edit, and redistribute the documentation, but you can not sell it, or sell any documentation derived from it. You can not modify or attempt to reverse engineer the programs.

This product is subject to U.S. export control laws and regulations including, but not limited to, the U.S. Export Administration Regulations, the International Traffic in Arms Regulation requirements, and all applicable end-use, end-user and destination restrictions. Licensee shall not permit, directly or indirectly, use of any Perforce technology in or by any U.S. embargoed country or otherwise in violation of any U.S. export control laws and regulations.

Perforce programs and documents are available from our Web site as is. No warranty or support is provided. Warranties and support, along with higher capacity servers, are sold by Perforce Software.

Perforce Software assumes no responsibility or liability for any errors or inaccuracies that might appear in this book. By downloading and using our programs and documents you agree to these terms.

Perforce and Inter-File Branching are trademarks of Perforce Software.

All other brands or product names are trademarks or registered trademarks of their respective companies or organizations.

Any additional software included within Perforce software is listed in License Statements on page 49.
Table of Contents

About This Manual .............................................................................................................. vii
  See also ............................................................................................................................. vii
  Please give us feedback ...................................................................................................... vii

Chapter 1  Getting Started with Insights ........................................................................... 1

Chapter 2  Installing and Upgrading Insights .................................................................... 3

  Installation options ........................................................................................................ 3
  Installing using the TGZ or ZIP file ................................................................................ 3
  Installing using the OVA ............................................................................................... 3
  Installing using the RPM or Debian package ................................................................ 5
    Prerequisites ............................................................................................................... 5
    Installation steps ........................................................................................................ 5
    Next steps .................................................................................................................. 7
  Upgrading ........................................................................................................................ 7
    Upgrading from a previous perforce-insights package installation using an RPM or Debian package ................................................................................................. 8
    Upgrading from a release-specific package installation using an RPM or Debian package .............................................................................................................. 8
    Upgrading from a previous TGZ or ZIP installation using an RPM or Debian package ............................................................................................................. 9
  Upgrading using the TGZ or ZIP archive ..................................................................... 10
  Installed files ................................................................................................................ 10

Chapter 3  Configuring Insights ....................................................................................... 11

  Configuring your PostgreSQL database ......................................................................... 11
    Enabling remote access and user/password authentication ........................................ 11
    Creating the database and role manually ................................................................... 13
    Testing database access ............................................................................................. 13
  Configuring Insights using the online administrative interface ..................................... 13
    Import page: importing a Perforce checkpoint and enabling continuous import .......... 14
    Setup page: setting configuration options .................................................................. 15
      Perforce Server options ......................................................................................... 15
      Jobs options .......................................................................................................... 16
    Projects page: configuring projects .......................................................................... 17
      Project Name .......................................................................................................... 18
      Private & Perforce Group Name ............................................................................ 18
      Job View ............................................................................................................... 18
      Branches .............................................................................................................. 18
      Users & Groups .................................................................................................... 18
      Generating Statistics ............................................................................................ 19
Chapter 4  Insights Plugin API  .................................................. 21

Writing a Plugin ....................................................................................... 21
The Web API ............................................................................................. 21
The Metric Class .......................................................................................... 23
getChartOptions() ...................................................................................... 25
getData() ................................................................................................... 26
Querying the Database .................................................................................. 27
Perforce to SQL ........................................................................................... 27
Interesting Tables .......................................................................................... 27
File Revisions .............................................................................................. 28
Submitted Changelists (p4change) ............................................................... 28
Projects and Files .......................................................................................... 29
Jobs and Fixes .............................................................................................. 29

Plugin to the Database ............................................................................... 31
Getting Data From the Database ................................................................. 31
Projects ........................................................................................................ 33
Complex Queries ........................................................................................... 33

Classes ............................................................................................................. 37
Chart ............................................................................................................... 37
Options ......................................................................................................... 38
Enums ........................................................................................................... 38
DataType ....................................................................................................... 38
Filter .............................................................................................................. 39
Fields .............................................................................................................. 39
DataWrapper ................................................................................................. 41
Fields .............................................................................................................. 41
Metric ............................................................................................................. 43
User request fields ......................................................................................... 43
Internal fields ................................................................................................. 43
Methods ......................................................................................................... 43
dailyTask() ................................................................................................. 43
dailyTask(Date startDate, Date endDate) ...................................................... 44
Chart getChartOptions() .............................................................................. 44
DataWrapper getData() .............................................................................. 44
String getMessage(String code) ................................................................... 44
String getMessageOrNull(string code) ............................................................ 44
List getSQL(MetricQuery query) ................................................................. 44
void updateSQL(MetricQuery query) ............................................................. 44
int getTaskOrder() ....................................................................................... 44
boolean hasTable(String tableName) ............................................................ 44

MetricQuery ................................................................................................. 45
Constructors ................................................................................................. 45
MetricQuery(String query) ........................................................................... 45
Methods ......................................................................................................... 45
void addDate(Date value) ............................................................................ 45
void addInt(int value) .................................................................................. 45
void addLong(long value) ........................................................................... 45
void addString(String value) ....................................................................... 45
List getParams() ......................................................................................... 45
String getQuery() ........................................................................................ 45

Data types .......................................................................................................................... 47
Granularity (enum) ..................................................................................................... 47
Methods ............................................................................................................. 47
String format(Date date) ............................................................................. 47
Date floor(Date date) .................................................................................. 47
Date ceiling(Date date) ............................................................................... 47
Date next(Date date) ................................................................................... 48

License Statements ............................................................................................................. 49
About This Manual

This guide tells you how to install and configure Perforce Insights. It also provides links to tutorial videos that show how to configure Insights and how to use the Insights user interface.

The installation and configuration chapters in this guide are intended for people responsible for installing, configuring, and maintaining Perforce Insights, and assume that you have at least intermediate-level knowledge of Perforce Server administration. These chapters cover tasks typically performed by a system administrator (for instance, installing and configuring the software and troubleshooting issues), as well as tasks performed by a Perforce administrator (like creating checkpoints and journals).

Depending on your site’s needs, your Perforce Insights administrator need not be your system administrator. And because Perforce requires no special system permissions, a Perforce Insights administrator does not typically require root-level access.

See also

For more information, see the following resources available at http://www.perforce.com:

• Perforce Server Administrator’s Guide: Fundamentals:

• Perforce training courses:
  http://www.perforce.com/instructor-led-training-overview
  http://www.perforce.com/support-services/elearning

• All Perforce documentation:
  http://www.perforce.com/documentation

• To obtain online help from within Perforce client programs:
  Command-line Client: Type `p4 help` from the command line.
  Graphical client applications: Click Help on the main menu bar.

• Consulting
  http://www.perforce.com/support-services/consulting-overview

• Technical Support
  http://www.perforce.com/support-services

Please give us feedback

Please send any comments or corrections to manual@perforce.com.
Perforce Insights is an analytics tool that presents vital information about how your projects, codelines, and teams are performing. Insights draws on data from the Perforce Server (P4D) and logs, displaying it in graphical form so that you can quickly understand what's going on. Rather than querying your Perforce service directly, Insights exports metadata from Perforce into a PostgreSQL database in real time and presents this data graphically using a web front end. Access control is via Perforce, and users can configure their own dashboards to control what they see.

To get started with the Insights web interface, watch this video tutorial:

To get started with installing and configuring Insights, watch this video tutorial:

For detailed information about installing and configuring Insights, see Chapter 2, “Installing and Upgrading Insights” on page 3 and Chapter 3, “Configuring Insights” on page 11.
Chapter 2  Installing and Upgrading Insights

This chapter discusses the following topics:

- Installation options
- Installing using the TGZ or ZIP file
- Installing using the OVA
- Installing using the RPM or Debian package
- Upgrading
- Installed files

Installation options

You can install Insights using any of the following methods:

- **RPM package**: provides a simplified way to install Insights on CentOS and Red Hat. An included script assists you with configuration. Your installation is updated with patches (and optionally major releases) automatically.

- **Debian package**: provides a simplified way to install Insights on an Ubuntu-based system. An included script assists you with configuration. Your installation is updated with patches (and optionally major releases) automatically.

- **Tarball (TGZ file)**: a compressed distribution archive for Linux installations.

- **ZIP file**: a compressed distribution archive for Windows installations.

- **OVA**: a virtual machine image in the Open Virtual Appliance (OVA) format. It includes everything you need to run Insights against your own Perforce Server instance, and is intended primarily for evaluation purposes.

The ZIP and TGZ files differ only in text file line endings.

Installing using the TGZ or ZIP file

This installation chapter covers only the OVA, RPM package, and Debian package installation instructions. For manual installations using the tarball or .zip archive, download the archive at [http://www.perforce.com/downloads/insights](http://www.perforce.com/downloads/insights) and see the included readme for instructions.

Installing using the OVA

Use the insights.ova virtual appliance when you want to do a quick installation for evaluation purposes. It includes an instance of Insights running in a Tomcat web container, an empty PostgreSQL database, and an ftp server for uploading database checkpoints from your Perforce service. You can install it using any virtualization framework that supports the import of OVA files.

1. Download the insights.ova:
2. Import the OVA into your virtualization framework.

Configure as required by your virtualization tool. Reinitialize the MAC address of all network cards if you are presented with the option.

3. Start the Insights virtual instance.

Note the URL of the Insights web application that is displayed once the virtual machine has finished starting up.

4. Configure Insights to connect to your Perforce service.

   a. Go to the URL of the Insights web application that is displayed once the virtual machine has finished starting up, logging in as insightsadmin with password P4insightsadmin.

   b. Go to the drop-down menu on the top right and select Setup.

      Configure the Perforce service connection options. See “Perforce Server options” on page 15.

   c. Restart the Insights web server.

      Go to the Insights virtual machine console and log in as user perforce with password perforce.

         $ sudo service tomcat6 restart

5. Import a Perforce checkpoint.

   a. Obtain a recent checkpoint from your Perforce service, or create a new one.

      If you do not know how to create a checkpoint, see the Perforce System Administrator’s Guide, "Supporting Perforce: Backup and Recovery."

   b. Upload the checkpoint to the Insights server.

      Go the Insights web interface, log in as insightsadmin/P4insightsadmin, and select Import from the drop-down menu on the top right.

      On the Import page, browse for the checkpoint file, select it, and click Upload Checkpoint.

         ![](Note) Alternatively, you can copy the checkpoint file directly to the ~/ftp directory on the Insights server. If you use ftp to copy the file, use username perforce and password perforce.

   c. On the Import page, select the checkpoint file from the Checkpoint Filename dropdown.

      Click Import.
Chapter 2. Installing and Upgrading Insights

Chapter 2. Installing and Upgrading Insights

d. Enable continuous import.

Continuous import reads changes from the Perforce service as they happen in real time, keeping your Insights data up to date.

For more information about importing checkpoints, see “Import page: importing a Perforce checkpoint and enabling continuous import” on page 14.

For more configuration options, see “Configuring Insights using the online administrative interface” on page 13 and “Setting up Perforce Service Manager” on page 20.

Installing using the RPM or Debian package

Prerequisites

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>See the Insights release notes for the most comprehensive and up-to-date software and hardware requirements.</td>
</tr>
</tbody>
</table>

- For the RPM package: Linux Intel x86_64 CentOS 6.x and Red Hat 6.x
- For the Debian package: Linux Intel x86_64 Ubuntu 12.04 ("Precise")
- Java JDK 6 or 7
- Tomcat 6 or 7
- PostgreSQL database 8 or 9, configured to use UTF-8
  - We recommend that you host your database on a different machine than Insights.
- Perforce Server 2012.1 or later
  - We recommend that you host Perforce Server on a different machine than Insights.
- Active internet connection

For the optional Perforce Service Manager (PSM) component:

- Ruby 1.8.7
- Rubygems

Installation steps

1. Import the Perforce package signing key.

As root, run one of the following:

For RPM:

```
rpm --import http://package.perforce.com/perforce.pubkey
```
For Debian:

```
wget -q http://package.perforce.com/perforce.pubkey -O- | sudo apt-key add -
```

For information about how to verify the authenticity of the signing key, see http://answers.perforce.com/articles/KB_Article/Public-Key-for-Installation-Packages.

2. Configure your package repository.

   • For RPM packages, create a file called `/etc/yum.repos.d/perforce.repo` with the following content:

```
[perforce]
name=Perforce
baseurl=http://package.perforce.com/yum/rhel/6/x86_64/
enabled=1
gpgcheck=1
```

This works for both Red Hat and CentOS.

   • For the Debian package, create a file called `/etc/apt/sources.list.d/perforce.sources.list` with the following line:

```
deb http://package.perforce.com/apt/ubuntu precise release
```

Run `sudo apt-get update` to update the package repository.

3. Install the Insights package.

There are two package files to choose from:

   • `perforce-insights`: a virtual package that installs the most recent stable version of Insights. Use this package to enable easy upgrades whenever a patch or new release is available. This package is recommended for most installation scenarios.

      For Centos and Red Hat, run:

      ```
yum install perforce-insights
      ```

      For Ubuntu, run:

      ```
      apt-get install perforce-insights
      ```

   • `perforce-insights-r15.1`: a direct installation of Perforce Insights 2015.1. You can run upgrade commands to get new patches but not new releases. Use this package if you are concerned that an automated update may accidentally upgrade Insights to a new release.

      For Centos and Red Hat, run:

      ```
yum install perforce-insights-r15.1
      ```
Chapter 2. Installing and Upgrading Insights

For Ubuntu, run:

```bash
apt-get install perforce-insights-r15.1
```

4. **Configure the PostgreSQL database.**

If you are running Ubuntu and your installation of PostgreSQL is on the same machine as Insights, you can skip this step. The `configure-insights.sh` script (Step 5) takes care of the Insights connection to the database.

For all other configurations (Ubuntu with PostgreSQL on a remote host; Red Hat or CentOS, whether PostgreSQL is local or remote), see Configuring your PostgreSQL database for detailed information about configuring PostgreSQL to work with Insights.

> **Important**
The PostgreSQL database must be configured to use UTF-8. For information about character set configuration in PostgreSQL, see [http://www.postgresql.org/docs/9.0/static/multibyte.html](http://www.postgresql.org/docs/9.0/static/multibyte.html)

5. **Configure the Insights installation using the `configure-insights.sh` script.**

The script takes you through the process of installing the Insights WAR file to the Tomcat application server and completing the configuration of the PostgreSQL database.

You can run the script interactively or non-interactively, the latter allowing you to script the installation by passing all of the options as command line arguments. Run the following for full details:

```bash
configure-insights.sh --help
```

**Next steps**

After you have run `configure-insights.sh`, you must:

1. Generate a Perforce database checkpoint.

2. Import the Perforce checkpoint and perform additional configuration options using the Insights online administrative interface.

3. Optionally, install and configure Perforce Service Manager (PSM) to monitor current Perforce Server performance.

For detailed instructions, see Configuring Insights

**Upgrading**

There are multiple upgrade paths, depending on your existing release version and installation method:

- To upgrade from Insights 13.2 to 15.1, you must first upgrade to 13.3. See the upgrade instructions in the *Perforce 13.3 Insights Administrator’s Guide*. 

• If you are upgrading from Insights 13.3 or 14.X to 15.1 and you installed 13.3 or 14.X using the perforce-insights RPM or Debian package, follow the instructions in “Upgrading from a previous perforce-insights package installation using an RPM or Debian package” on page 8.

• If you are upgrading from Insights 13.3 or 14.X to 15.1 and you installed 13.3 or 14.X using the perforce-insights-rnn.n RPM or Debian package, follow the instructions in “Upgrading from a release-specific package installation using an RPM or Debian package” on page 8.

• If you want to upgrade from Insights 13.3 to 15.1 using an RPM or Debian package and you installed 13.3 or 14.1 using the TGZ or ZIP archive, follow the instructions in “Upgrading from a previous TGZ or ZIP installation using an RPM or Debian package” on page 9.

• To upgrade using the TGZ or ZIP archive, see the readme file included with the archive.

**Upgrading from a previous perforce-insights package installation using an RPM or Debian package**

If you installed Insights 13.3 or 14.X using the perforce-insights package, use the following steps to upgrade to 15.1.

1. **Shut down and back up your current Insights instance.**
   - Shut down the Tomcat instance in which the old Insights is running.
   - Back up the PostgreSQL database.

2. **Update the Insights package.**
   
   For Centos and Red Hat, run:
   
   ```shell
   yum update perforce-insights
   ```

   For Ubuntu, run:
   
   ```shell
   apt-get update
   apt-get dist-upgrade perforce-insights
   ```

3. **Run configure-insights.sh to configure your Insights installation.**
   
   The script reads your existing configuration properties and applies them to your new installation.

4. **Restart the Insights Tomcat instance.**
   
   ```shell
   service tomcat6 restart
   ```

**Upgrading from a release-specific package installation using an RPM or Debian package**

If you installed the previous release using the perforce-insights-rnn.n package, use the following steps to upgrade to 15.1:
1. **Shut down and back up your current Insights instance.**
   - Shut down the Tomcat instance in which the old Insights is running.
   - Back up the PostgreSQL database.

2. **Install the new Insights package.**
   There are two package files to choose from:
   - **perforce-insights**: a virtual package that installs the most recent stable version of Insights. Use this package to enable easy upgrades whenever a patch or new release is available. This package is recommended for most installation scenarios.

   For Centos and Red Hat, run:
   ```bash
   yum install perforce-insights
   ```

   For Ubuntu, run:
   ```bash
   apt-get update
   apt-get install perforce-insights
   ```

   - **perforce-insights-r15.1**: a direct installation of Perforce Insights 2015.1. You can run upgrade commands to get new patches but not new releases. Use this package if you are concerned that an automated update may accidentally upgrade Insights to a new release.

   For Centos and Red Hat, run:
   ```bash
   yum install perforce-insights-r15.1
   ```

   For Ubuntu, run:
   ```bash
   apt-get update
   apt-get install perforce-insights-r15.1
   ```

3. **Run `configure-insights.sh` to configure your Insights installation.**
   The script reads your existing configuration properties and applies them to your new installation.

4. **Restart the Insights Tomcat instance.**
   ```bash
   service tomcat6 restart
   ```

**Upgrading from a previous TGZ or ZIP installation using an RPM or Debian package**

If you installed the previous release manually using the tarball (TGZ) or ZIP archive and you want to upgrade using the Deb or RPM package, follow the instructions in “Installing using the RPM or Debian package” on page 5.
When you run `configure-insights.sh`, answer `Y` (yes) when the script asks “Do you want to overwrite any existing configuration? (y/n)?” This enables the script to remove the old version of the Insights WAR file and reconfigures your Insights instance to use the package layout. The script will keep your existing configuration options.

**Upgrading using the TGZ or ZIP archive**

For information about upgrading using the tarball (TGZ) or ZIP archive, see the `readme` file included with the archive.

**Installed files**

Once the Debian or RPM package is installed, its file system structure looks like this:

```
/usr/
  sbin/
    configure-insights.sh -> /opt/perforce/usr/sbin/configure-insights.sh
/opt/
  perforce/
    usr/
      sbin/
        configure-insights.sh
    share/
      insights/
        db/
          create_pa_tables.sql
          create_template_user.sql
          upgrade_13_3.sql
          upgrade_14_2.sql
          upgrade_15_1.sql
          insights-config.properties
          insights-log4j.groovy
          insights.war
      plugins/
```

After `configure-insights.sh` has been run, a symbolic link is created from the Tomcat `webapp` directory to the `insights.war` in `/opt/perforce`.

**Note**

The Perforce command line interface package, which is installed as a part of the `perforce-insights` and `perforce-insights-rnn.n` packages, puts the `p4` executable in `/opt/perforce/usr/bin`. 
This chapter discusses the following configuration tasks:

- **Configuring your PostgreSQL database** (required for most installations)
- **Configuring Insights using the online administrative interface** (required for all installations)
- **Setting up logging** (optional)
- **Setting up Perforce Service Manager** (optional)

### Configuring your PostgreSQL database

By default, an installation of PostgreSQL usually requires a valid operating system account to connect to a database. Because the Insights database connection comes from a Tomcat web application, Insights cannot access the database using a system account, but must instead connect using username/password authentication. If this has not been enabled already, you must configure the database to enable it.

If you are running Ubuntu and your installation of PostgreSQL is on the same machine as Insights, you can skip this configuration. For all other configurations (Ubuntu with PostgreSQL on a remote host; Red Hat or CentOS, whether PostgreSQL is local or remote), you must ensure that the database can be accessed with a username and password. In addition, if you do not want to provide remote administrator access to the database, you must create the role and database manually.

**Important**

The PostgreSQL database must be configured to use UTF-8. For information about character set configuration in PostgreSQL, see [http://www.postgresql.org/docs/9.0/static/multibyte.html](http://www.postgresql.org/docs/9.0/static/multibyte.html)

### Enabling remote access and user/password authentication

If you are running Insights and PostgreSQL on different machines, you must enable user/password authentication for remote access to the PostgreSQL database. You must also enable user/password authentication when PostgreSQL is local if you are running Red Hat or CentOS.

For more information about setting up PostgreSQL authentication, see [http://www.postgresql.org/docs/8.4/static/auth-methods.html](http://www.postgresql.org/docs/8.4/static/auth-methods.html).

To enable remote access and password authentication:

1. **If you are running Insights and PostgreSQL on different machines, configure the database to allow access from remote clients.**

   For CentOS or Red Hat, edit the following file:
   ```bash
   /var/lib/pgsql/data/postgresql.conf
   ```
   
   For Ubuntu, edit the following file (Ubuntu 12.04 uses PostgreSQL 9.1):
   ```bash
   /etc/postgresql/9.1/main/postgresql.conf
   ```
Configure **listen_addresses** to allow access from the Insights application server. The following configuration example opens up the database to clients from any host, but you may want to restrict the configuration to a narrower definition:

```
listen_addresses = '*'
```

2. **Enable password authentication.**

For CentOS or Red Hat, edit the following file:

```
/var/lib/pgsql/data/pg_hba.conf
```

For Ubuntu, edit the following file (Ubuntu 12.04 uses PostgreSQL 9.1):

```
/etc/postgresql/9.1/main/pg_hba.conf
```

At the bottom of this file are a number of entries ending with `ident` (in PostgreSQL 9.1, these entries may end with `peer`). The default PostgreSQL configuration looks something like this:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DATABASE</th>
<th>USER</th>
<th>CIDR-ADDRESS</th>
<th>METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>all</td>
<td>all</td>
<td></td>
<td>ident</td>
</tr>
<tr>
<td>IPv4</td>
<td>all</td>
<td>all</td>
<td>127.0.0.1/32</td>
<td>ident</td>
</tr>
<tr>
<td>IPv6</td>
<td>all</td>
<td>all</td>
<td>::1/128</td>
<td>ident</td>
</tr>
</tbody>
</table>

The simplest solution is to change each instance of `ident` (or `peer`) to `md5`.

You may also want to open up the access rules for one or both of the `host` lines (depending on whether your network infrastructure uses IPv4 or IPv6). For example, to allow password access over all network types and to open up host access from all IPv4 addresses, you can use a configuration like the following:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DATABASE</th>
<th>USER</th>
<th>CIDR-ADDRESS</th>
<th>METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>all</td>
<td>all</td>
<td></td>
<td>md5</td>
</tr>
<tr>
<td>IPv4</td>
<td>all</td>
<td>all</td>
<td>0.0.0.0/0</td>
<td>md5</td>
</tr>
<tr>
<td>IPv6</td>
<td>all</td>
<td>all</td>
<td>::1/128</td>
<td>md5</td>
</tr>
</tbody>
</table>

If you want to limit access further, consult the PostgreSQL documentation.

Note that by default, the **postgres** user does not have a password set, and you cannot log in as this user with a password until you have enabled password authentication. Until you do, you can still log in by using the postgres user account:

```
sudo -u postgres psql
```
Creating the database and role manually

If the database host is remote and you do not want to give remote administrator access to the database, then the database administrator must create the role and database manually:

1. **Create a role and give it a password.**

   Insights expects the role name `insights` by default, but you can use any role name you like and tell Insights during configuration.

2. **Create a database.**

   Insights expects the name `insightsdb` with owner `insights`, but you can use any name you like and tell Insights during configuration.

3. **Run the installed SQL files against the new database.**

   The `perforce-insights` Deb and RPM package installs the SQL files into `/opt/perforce/usr/share/insights/db`. If you are installing manually from the .zip or .tgz file, the SQL files are in the `sql` directory.

   a. These files create tables with owner `insights`. Rename the script owner if you created a different role name.

   b. Run the `create_pa_tables.sql` script

   c. Run the `create_template_user.sql` script

When you run the `configure-insights.sh` script on the Insights application server, you will be asked if you want to create a database. Say no and enter the role name and password you set up here.

Testing database access

To test access to the database, run the following:

```bash
psql -W -U insights -h localhost -d insightsdb -c "SELECT 1"
```

Replace `insights` and `insightsdb` with your role and database name. Replace `localhost` with the hostname of the server if the database is remote. This command should return a single row, if access is enabled for the user `insights` against the database `insightsdb`.

Configuring Insights using the online administrative interface

Insights provides an administrative interface that you can use to import a Perforce database checkpoint, update configuration options, set up projects, and manage users.

The first time you log in, you can use insightsadmin with a password of P4insightsadmin.
This is the default Insights administrative account, which allows you to configure the system.

2. Go to the drop-down menu on the top right.
   There are options for **Setup**, **Projects**, **Import**, and **Users**. Each of these options is described below.

### Import page: importing a Perforce checkpoint and enabling continuous import

The first time you install and set up Insights, you must import a Perforce database checkpoint.

1. **Obtain a recent checkpoint from your Perforce Server, or create a new one.**
   If you do not know how to create a checkpoint, see the *Perforce System Administrator’s Guide*, "Supporting Perforce: Backup and Recovery."

2. **Upload the checkpoint to the Insights server.**
   Go the Insights web interface, log in as an Insights administrator (the default is insightsadmin/P4insightsadmin), and select **Import** from the drop-down menu on the top right.
   On the **Import** page, browse for the checkpoint file, select it, and click **Upload Checkpoint**.
   Alternatively, you can copy the checkpoint file directly to the Insights server using a mechanism like ftp, scp, or network shares. You must copy the file to the directory defined in the configuration option **insights.upload.path**. The default value for this option depends on your operating system. For more information about setting configuration options, see "**Setup page: setting configuration options**" on page 15.

3. **On the Import page, select the checkpoint file from the Checkpoint Filename dropdown.**
   Click **Import**.
   Verify that the checkpoint character set encoding is correct. If your Perforce service runs in Unicode-mode, Insights detects the character set and you can ignore this setting.
   Click Import.

4. **Enable continuous import.**
   Continuous import reads changes from the Perforce service as they happen in real time, keeping your Insights data up to date.
   Insights needs to know where to stream changes from, so before this can happen you should configure the **insights.p4port**, **insights.user** and **insights.password** configuration options on the **Setup** page (see below).
   The Import process reads any changes to the Perforce database that occur between the time you generate the checkpoint and the point at which continuous import begins, as long as there are journals available for the period.
Do not use the -z option with `p4d -jc` or `p4d -jj` to compress checkpoints and journals. Instead use the -Z option so that journals remain uncompressed. Insights uses the `p4 export` command to extract data from the Perforce database journal. In some cases the journal can be rotated before `p4 export` has finished extracting the data. It is important therefore that any rotated journal is left uncompressed to enable `p4 export` to continue.

The location of the rotated journal must be known to the Perforce service. The `p4 export` command expects the journal to be in the P4D server root (`P4ROOT`), unless you specify a directory with the Perforce Server configurable `journalPrefix`. If your Perforce Server configuration stores journals outside of the `P4ROOT` directory (which is recommended practice), then you must use the `journalPrefix` server configurable. If your journals are being moved to a different location after being rotated, or if a journal prefix is being specified on the command line, `p4 export` will not be able to find them.

For more information about configuring journal compression, see the *Perforce System Administrator's Guide*, "Supporting Perforce: Backup and Recovery".

The Import page also displays current import status, with the progress bar showing how far through the checkpoint import the process has gone, along with the journal number and offset reached. You can enable more detailed DBImport logging using `insight-config.properties`. For more information, see Setting up logging.

### Setup page: setting configuration options

The Setup page displays a list of all of the configuration options for Insights. Not all of them are described here - most can be left unchanged.

- To edit an option, click the option name and select **Edit**.
- To delete an option, click the option name and select **Delete**.
- To create a new option, click **New Configuration**.

**Note**

Initially, the values created in `pa_insights_configuration` are based on properties (for example `/opt/perforce/usr/share/insights/insights-config.properties`). Once a value is stored in the database on startup, modifying it in the properties file has no effect, because the application uses the values in the database table. Therefore, after initial startup you should use the Setup page to add, update, or remove configuration values.

### Perforce Server options

You must define the Perforce Server instance that will be used for login authentication and continuous import. Until you set the Perforce Server information, non-administrative users will be unable to log into Insights using their Perforce user name and password.

**Important**

You must restart the Tomcat application server after changing any of these options.
• **insights.p4port**: P4PORT setting for the Perforce server, such as `ssl:myserver:1666`.

• **insights.user**: The username used to authenticate against the Perforce Server. This must be a super user. We recommend that it be a service user.

• **insights.password**: the password or ticket for the Perforce user. For security reasons, we recommend that you define a long-lived ticket for this user and enter the ticket value here rather than the password.

Instead of entering the ticket value, you can place a ticket file in the home directory of the account running Tomcat. Typically for Tomcat 6 on Linux, this is `/usr/share/tomcat6`. The ticket file is called `.p4tickets` and is created in your home directory when you log into the Perforce service.

For more information about creating long-lived tickets for service users, see the section on service users in *Distributing Perforce*, "Perforce Replication" ([http://www.perforce.com/perforce/doc.current/manuals/p4dist/chapter.replication.html#replication.serviceusers](http://www.perforce.com/perforce/doc.current/manuals/p4dist/chapter.replication.html#replication.serviceusers)).

• **insights.psm.baseURL.main**: If you are using the optional Perforce Service Manager (PSM) component, you must also set the URL of the PSM web service, using the format `http://hostname:port/api`.

For more information, see [Setting up Perforce Service Manager](#).

### Jobs options

If you are using Perforce jobs, you must specify how to interpret the Perforce jobspec. Since a jobspec can vary widely in how it is structured, and there are no limitations on how the fields are used, you must configure the names of the job fields and their possible values. If you do not have experience with Perforce job views, see the *Perforce System Administrator’s Guide*, "Customizing Perforce: Job Specifications".

• **insights.jobs.bugs.field.name**: The name of the jobspec field that defines the type of job (for example, bug, defect, task, story). Defaults to 'Type'.

• **insights.jobs.bugs.field.value**: The value of the Type field named above that defines a job as being a bug. Defaults to 'BUG'.

• **insights.jobs.bugs.severity.name**: The name of the jobspec field that defines the severity (or importance) of a bug. Defaults to 'Severity'.

• **insights.jobs.bugs.status.name**: The name of the jobspec field that defines the status of a job (e.g. open, closed, duplicate). Defaults to 'Status'.

• **insights.jobs.bugs.status.open.value**: Values for the 'Status' field that should be interpreted as open. This can contain multiple comma separated values. Defaults to 'open'.

• **insights.jobs.bugs.status.fixed.value**: Values for the 'Status' field that should be interpreted as closed. This can contain multiple comma separated values. Defaults to 'fixed,closed,punted,obsolete,duplicate'.

• **insights.jobs.bugs.date.reported**: The name of the jobspec field that holds the date a job was first opened. Defaults to 'ReportedDate'.
• **insights.jobs.bugs.date.modified**: The name of the jobspec field that holds the date a job was last modified. Defaults to 'ModifiedDate'.

The following properties enable you to specify how job severities are displayed in the graphs. By default, there are three severities configured, and each has two settings. These are:

• **insights.jobs.bugs.values.A.weight** = 10
• **insights.jobs.bugs.values.A.color** = red
• **insights.jobs.bugs.values.B.weight** = 3
• **insights.jobs.bugs.values.B.color** = #ff5105
• **insights.jobs.bugs.values.C.weight** = 1
• **insights.jobs.bugs.values.C.color** = orange

These settings give severity A bugs a weight of 10, severity B bugs a weight of 3, and a severity C bug a weight of 1. They also configure the colors used for these severities when they are displayed on the Open Bugs bar chart.

To change the weights or colors, simply edit these values.

If you do not use A, B, and C for your severities, create new settings for the severities that you do use and delete the existing ones, if necessary. For example, if your severities are CRITICAL, HIGH, MEDIUM, and LOW, then:

• **insights.jobs.bugs.values.CRITICAL.weight** = 12
• **insights.jobs.bugs.values.CRITICAL.color** = #770000
• **insights.jobs.bugs.values.HIGH.weight** = 6
• **insights.jobs.bugs.values.HIGH.color** = #ff0000
• **insights.jobs.bugs.values.MEDIUM.weight** = 3
• **insights.jobs.bugs.values.MEDIUM.color** = #ff5105
• **insights.jobs.bugs.values.LOW.weight** = 1
• **insights.jobs.bugs.values.LOW.color** = orange

If you do not use Perforce Jobs, then Insights will not be able to show you information about bugs and issues. If you use a third party defect tracking tool, such as JIRA, then it is possible to use P4DTG to import defect information into the Perforce Server, which would allow Insights to report on them. For more information on P4DTG see [http://www.perforce.com/product/components/defect-tracking-gateway](http://www.perforce.com/product/components/defect-tracking-gateway)

**Projects page: configuring projects**

Initially, Insights does not know how the files in your depot are organized. In order to tell Insights how you define projects in Perforce, log in as **insightsadmin** and select **Projects** from the menu at the top right. You will see a list of all currently configured projects. Initially, the list will be empty.
Note
Any user assigned the ROLE_PROJMGR role on the User Management page can configure projects. For more information, see User Management page: assigning administrators and project managers.

Configuring projects in Perforce Insights involves giving each project a name and identifying the branches and depot paths included in the project.

Select Add Project for each project, and fill in the details.

**Project Name**
This is a descriptive name for the project.

**Private & Perforce Group Name**
Normally, information about projects can be accessed by anyone who can log in to Insights. If a project needs restricted access, check this box.

A private project can only be accessed by logged-in users who are members of a specific Perforce group. The name of this group is automatically generated from the project name and cannot be set manually. If your project is called "My Project," then the group name is insights_project_my_project.

Users who are not members of this group will not be able to select this project from the various filters. However, they will not be restricted from viewing parts of the depot within the project through other means.

**Job View**
This is the job query that should be used to find jobs (bugs) for this project. The nature of this query depends on how the jobspec is defined. If there is a field named project that holds the name of the project that a bug is assigned to, then the query might look like this:

```
project=projectname
```

For a complete discussion of how to write job queries, see the *P4 User’s Guide, "Defect Tracking".*

**Branches**
The branches of a project define the depot paths that include files associated with the project. A project can have multiple branches, such as MAIN, DEV, or RELEASE, and each branch consists of one or more depot paths.

Add a branch, give it a name, and specify each depot path that includes files for that part of the project.

**Users & Groups**
Each project can have users associated with it. User filters for a project are limited to those users whom you define as members of the project. If a user is not associated with a project, then you will not be able to filter by that user, though data about them may still be displayed.
If you want to include a large number of users, it is suggested that you set up a group in the Perforce service and add that to the project.

**Generating Statistics**

On the main **Projects** page, there is a link on the bottom right-hand corner to generate statistics for projects. Once a project has been created, a process runs every night to generate statistics for that project. However, when you first create a project, you must run that process for historical data.

The Statistics Generation page enables you to select the projects for which you want to generate historical data and the time period you want to cover. The greater the time period, the longer it takes.

The time required to generate statistics from historical data depends very much on the amount of that data (number of files, submits, and jobs) in each project. We recommend that you initially generate data for six months for a single project, and then base your estimates for the rest of data generation on the results.

Regenerating data for longer time periods for a project can be done at any time.

**User Management page: assigning administrators and project managers**

Insights comes with two internal users:

- **insightsadmin**: the default administrative user.
- **insights_template_user**: a user whose dashboard preferences are used as a default to populate the Insights dashboards and widgets the first time a user logs into Insights.

You can configure a different user as the user whose dashboards and widgets are copied as the default for first-time logins. On the **Setup page**, set the configuration property `insights.configuration.copyDashboardsFromUser` to the user whose dashboards and widgets you want to copy as the default for new users.

The Insights User Management page enables you to change the password for **insightsadmin** and **insights_template_user** and set up other users to be administrators or project managers.

- **Note**: You cannot use this interface to change the password of users authenticated by Perforce, only the internal users **insightsadmin** and **insights_template_user**.

To access the **Insights User Management** page, log in as **insightsadmin** and select **Users** from the menu at the top right.

To find a user, start typing their name into the **Username** field. Only users who have previously logged into Insights or who were created using the User Management interface are listed.

To modify a user that has not previously logged in, use the **Create** option (in the toolbar). Creating users using this option does not create them in Perforce and does not allow you to set their password.

Once a user has been created, you can edit the user’s role, using the **Roles** tab on the **Create User** page. A user that has the **ROLE_ADMIN** role has permission to configure Insights (they see the Setup option on their menu). Users with **ROLE_PROJMGR** have permission to modify Projects within Insights. They see the Projects option on their menu.
Setting up logging

Insights uses Apache log4j for logging. The default log location is /var/log/tomcatn/insights.log.

You can change the logging level and path in /opt/perforce/usr/share/insights/insights-config.properties.

For advanced logging configuration, you can use the sample log configuration file insights-log4j.groovy. Contact Perforce technical support for instructions about how to use this file.

The Import page displays basic DBImport logging details. For more detailed DBImport logging, change the logging level in insight-config.properties.

Setting up Perforce Service Manager

Perforce Service Manager (PSM) is an optional component of Insights that monitors the current performance of a Perforce Server instance. It reports on the current command queue length and shows the most active users and processes using Perforce.

The PSM is a standalone component that you must install on the host that runs your Perforce Server, since it monitors the log files from the Perforce Server instance in real time. PSM downloads, installation instructions, and configuration instructions are available on your Insights server here:

http://hostname:port/insights/docs/psm

Once you have set up PSM, you should ensure that the configuration option insights.psm.baseURL.main on the Setup page has been configured correctly.
Chapter 4  Insights Plugin API

This chapter describes the API for Insights plugin development, and provides a guide to help you develop your own plugins. As a start, sample plugins are included as part of the Insights installation, and can be found in the `webapps/plugins` directory.

**Writing a Plugin**

Perforce Insights already provides a number of metrics which can be used to determine the state of your projects held within your Perforce server, but there are plenty more which could be written. Insights has been designed to be extensible to allow anyone to write their own metrics and plug them into the system for use by users.

An Insights plugin is a Groovy class that follows a certain pattern:

```groovy
class ExampleMetric extends Metric {
    public Chart getChartOptions();
    public DataWrapper getData();
}
```

The `getChartOptions()` method returns an object that describes what the graph looks like on the Insights dashboard, so is the same every time the metric is displayed. The `getData()` method returns the data itself, so will vary depending on the current filter settings and the state of the data in the database.

However, before diving into the details of how to write your own metrics, we shall first take a look at how metrics are called and used.

**The Web API**

Insights is a web application which makes extensive use of Web APIs to provide functionality to pretty much all aspects of the front end UI. Each of these APIs is REST based, and returns data in JSON format, making it easy to process using Javascript.

In order to access the APIs, you first need to be logged into Insights. You can then access each API directly from your browser by querying the server like you would a typical web page. The only difference is that the returned result is Javascript (JSON) data rather than HTML.

For example, to get a list of all the metrics currently installed on Insights you can make a call to the following URL:

```
http://insights.local:8080/insights/api/metrics/list
```

You will need to replace `insights.local:8080` with the server where your instance of Insights is installed. What will be returned is a list similar to the following (for a vanilla installation of Insights):
Chapter 4. Insights Plugin API

Each item listed is the name of a metric, which is a unique string used to identify it when making a call directly to that metric.

For example, it’s possible to call a metric directly via the `/data` API. The following will get a list of the most active files between the two given dates:

```
http://insights.local:8080/insights/api/metrics/data/mostActive?
startDate=2014/01/01&endDate=2014/03/01
```

This calls the `mostActive` metric which returns the most edited files as data in JSON format:

```
{
  "title": null,
  "xtitle": "1 Jan 2014 - 31 Mar 2014",
  "ytitles": {
    "root": "Times Edited"
  },
  "xlabels": [
    "//depot/tools/build/jenkins-configs/config.xml",
    "//depot/intranet/Eval-Demo/RFP/FOO/20140104/Presentation.pptx",
    "//depot/intranet/qa/metrics/BAR/Reports/p4cms.html",
    "//depot/intranet/qa/metrics/BAR/Reports/p4sandbox.html",
    "//depot/intranet/qa/metrics/Burndown/Reports/commons.html",
    "//depot/intranet/qa/metrics/OpenBUGs/Daily_Reports/daily_open_all.out",
    "//depot/intranet/qa/metrics/OpenBUGs/Daily_Reports/daily_open.out",
    "//depot/intranet/qa/metrics/OpenBUGs/Daily_Reports/daily_open_all.out",
    "//depot/intranet/qa/metrics/OpenBUGs/Daily_Reports/daily_open.out",
    "//depot/intranet/qa/metrics/OpenBUGs/Daily_Reports/daily_open_all.out",
    "//depot/intranet/qa/metrics/OpenBUGs/Daily_Reports/daily_open.out"
  ],
  "series": null,
  "data": [ 37, 30, 28, 28, 28, 28, 25, 23, 21, 21 ],
  "ylimits": null
}
```

Within the Insights dashboard, this data would be displayed as a bar chart with ten columns, one for each of the listed files. The file names are the labels on the X axis (the `xlabels` property) and the values are held in the `data` array.

Data can be returned as ‘data’ or as a ‘series’. Whilst ‘data’ is an array of values, ‘series’ would be an array of an array of values, and is used when we have stacked graphs with multiple sets of values.

The `/data` API call is probably the most important, since it is the one that fetches information into the charts. In our example we pass two parameters to it, `startDate` and `endDate`. These specify the date range that the metrics should be returned for, and must both be in the format `yyyy/MM/dd`.
At this point we should probably clarify what we mean by a metric. In the Insights web front end, we call the charts that are displayed widgets, not metrics. So what’s the difference? A metric defines the type of query that is run against the database, as well as the type of filters that can be passed to the query.

A widget is the chart on an Insights dashboard. Each widget is backed by a metric, but also defines values for all the filters that are passed to it. For example, a "Top Contributors" metric might run a query to return a list of the most prolific contributors over a given period for a given project. The widget defines what the project is, and the date range to use. You could have several "Top Contributor" widgets, each for a different project, all calling the same metric at the back end.

There is a common set of filter values which can be supported by each metric. Some aren’t supported because they don’t make sense, others aren’t supported for simplicity. It is up to a metric to implement how each filter is used.

From the point of view of the REST API, the filters are the parameters passed to the /data API. In the above example, startDate and endDate were the two filters.

Some of the other filters supported are as follows:

- **granularity**
  - If set to one of DAYS, WEEKS or MONTHS, then data will be returned at that granularity. If the date range covers a 3 month period, then DAYS will return about 90 data points, WEEKS about a dozen and MONTHS will return three. You can also set it to be RANGE, in which case a single data point will be returned for the sum of the entire range.

- **projects**
  - A list of projects to filter by. Projects are identified by their numerical id, and given as a comma-separated list.

- **projectid**
  - A single project. Some metrics can only display data on a single project. These ones take a projectid filter rather than a projects filter in order to make this limitation explicit.

- **limit**
  - Limit the number of returned results. Used for metrics which return the ‘top N’ results.

### The Metric Class

An Insights plugin is written as a Groovy class, the simplest definition of which looks something like this:

```groovy
class HelloWorldMetric extends Metric {
    public Chart getChartOptions();
    public DataWrapper getData();
}
```

Groovy is a language based heavily on Java, that runs on the Java Virtual Machine. If you are not familiar with Groovy, but are with Java, then it will look very similar. The main differences are that Groovy is more dynamic, and allows a number of syntactic short cuts to make code easier to write.

To explain how a plugin is implemented, we will take a look at the HelloWorld metric that ships as an example with Insights.
You can test this plugin by installing it yourself on your own instance. To do this, copy the `HelloWorldMetric.groovy` and `helloWorld.properties` files out of the installation package and into the Insights plugin directory.

If you have installed Insights from a Linux package, then the example plugins will be in the `/opt/perforce/usr/share/insights/plugins` directory.

If you have installed Insights from either the `.tgz` or `.zip` file, then the examples can be found in the `webapps/plugins` directory.

The plugin directory defaults to `/var/lib/tomcat6/shared/plugins`, so this will need to be changed if you are on Windows. Login to Insights as the `insightsadmin` user, and navigate to the Setup menu. The current location of the plugins directory is defined by the `insights.metrics.plugindir` option, which can be set to a suitable value. On Windows, the directory path should use `/` rather than `\` as a path separator.

Once you have checked that this directory exists, and that it can be read by Tomcat, copy the `HelloWorldMetric.groovy` and `helloWorld.properties` files into it (again ensuring that they can be read by Tomcat).

For example, if you are running Tomcat 6 on Linux, you might create the plugin directory and set its permissions by running the following commands as root:

```
mkdir -p /var/lib/tomcat6/shared/plugins
chown tomcat6:tomcat6 /var/lib/tomcat6/shared/plugins
chmod 750 /var/lib/tomcat6/shared/plugins
```

If you are using a location other than `/var/lib/tomcat6/shared/plugins` or your instance of Tomcat runs as a different user, then you will need to modify these commands. If you are on Microsoft Windows, then you probably don’t need to worry about file permissions.

You should now copy the two files, `HelloWorldMetric.groovy` and `helloWorld.properties` into this directory.

You now have two choices. You can either restart Tomcat, or, you can tell Insights to refresh its cache of plugins. To do the latter, whilst you are logged in as the `insightsadmin` user, go to the following URL:

```
http://server:port/insights/api/metrics/refresh
```

Where `server:port` is the server and port of your Insights installation. This will force Insights to clear its cache of plugins, and re-read them from disc next time they are needed. You can check to see what plugins are known by pointing your browser at the following URL:

```
http://server:port/insights/api/metrics/list
```

If the new plugin has been found, this should return a list that includes `helloWorld` at the end, similar to the following:
The `helloWorld` metric should now be listed. The name of the metric is based on the class name defined in the plugin. Since Groovy is Java-based, the class name should match the filename. In this case we have named the class `HelloWorldMetric`, so the Groovy file is named `HelloWorldMetric.groovy`. The metric is then automatically named `helloWorld` by Insights.

To test it, go back into the Insights UI, and go to either the Overview or Hot Spots dashboards (or, if you don’t have these, use a Multi-Project dashboard). When you add a widget, there should be the option to display the Hello World widget.

It's not a very exciting widget, since it only displays static data, but that makes it easy to explain what it's doing. The two methods that are implemented by `HelloWorldMetric` are described next.

**getChartOptions()**

The first method that must be implemented is `getChartOptions()`, which returns a `Chart` object that describes what a displayed chart looks like on the dashboard. The method consists entirely of setting options, with no logic. The method is called once, when the plugin is first read, and then the data is cached, so this method does not have access to any of the filter parameters that might be passed by an individual widget at 'run time'.

```java
public Chart getChartOptions() {
    Chart chart = new Chart()
    chart.seriesGroups.put("root", [ "type": "column", "value": "number" ])
    chart.options.put("basic",
                        [ "stacked": false, "fill": true, "legend": false, "zoom": false ])
    chart.options.put("thumbnail", [ "cursor": false ])
    chart.options.put("expanded", [ "cursor": true ])
    chart.dataType = Chart.DataType.RANGE
    chart.filters.add(Chart.Filter.PROJECTS)
    chart.filters.add(Chart.Filter.DATERANGE)
    return chart
}
```

There are four things that need to be set on the `chart` object: the `seriesGroups`, the `options`, the `dataType` and the `filters`.

```java
chart.seriesGroups.put("root", [ "type": "column", "value": "number" ])
```
This tells Insights that we want a column graph (vertical bars for each data point) rather than a line graph, and that we want to display raw numbers rather than percentage values. You can probably guess from this description that column could be changed to line, and value to percentage.

The use of root also tells Insights something, but more of that later.

```javascript
chart.options.put("basic",
    [ "stacked": false, "fill": true, "legend": false, "zoom": false ])
```

Here we set the 'basic' options on the graph. This tells Insights that the graph should be filled (the usual option for bar graphs), and that it doesn't have a legend or a zoom option. The stacked setting means that there is only one value per data point. More on stacked graphs later.

```javascript
chart.options.put("thumbnail", [ "cursor": false ])
chart.options.put("expanded", [ "cursor": true ])
```

We can also set options specifically for the thumbnail (the small chart that is shown on the dashboard) and the expanded (full screen) versions of the chart. For example, we could enable zoom and the legend on the full screen version if we wanted to.

```javascript
chart.dataType = Chart.DataType.RANGE
```

This controls how dates are managed. For the helloWorld example, we aren't splitting data by date, so we use RANGE (the values are the same over the entire date range). Alternatively, we could set this to be PERIOD.

```javascript
chart.filters.add(Chart.Filter.PROJECTS)
chart.filters.add(Chart.Filter.DATERANGE)
```

Finally, we set the filters that can be applied to this metric. In this case it takes a list of projects and a date range. These aren't actually used by this particular plugin (because it just outputs static data), but are needed in order to determine which type of dashboard the metric will be listed on.

There are three dashboard types: Multi-Project, Single-Project and Server. By listing these filters, we ensure that it appears on the Multi-Project dashboards.

### getData()

With the presentation layer out of the way, we can take a look at the data side of things. The `getData()` method returns a `DataWrapper` object which defines all the labels and chart data that is placed into the chart.

In our helloWorld example, it looks like this:
public DataWrapper getData() {
  DataWrapper wrapper = new DataWrapper()

  wrapper.title = getMessage("title")

  // Example column chart.
  wrapper.ytitles = [ "0": "Greetings" ]
  wrapper.xtitle = "Greeters"

  wrapper.xlabels = [ "Alice", "Bob", "Eve" ]
  wrapper.data = [ 17, 13, 23 ]

  return wrapper
}

The most important two lines are the ones setting xlabels and data, they set the labels and values for the three data points on this graph. They are both arrays, and the order and number of elements in each array is important. The first element in the label matches the first element in the data, and so on.

A plugin allows I18N properties to be read from properties files located in the plugin directory. They are named after the metric's name, so in this case text is read from `helloWorld.properties`. If you wanted to support a German translation, you can do so by adding a `helloWorld_de.properties` file.

Something this plugin doesn't do is process any of the filter information. If we wanted to change the data shown based on the project selected in the widget then we would do this within the `getData()` function. How to access this information is described later.

### Querying the Database

Our `HelloWorldMetric` example was very simple, but displaying a graph of static data isn't particularly useful. The point of Insights is to render data on your Perforce system, which means that we need to start pulling data out of the database. This means understanding the data that is stored there.

#### Perforce to SQL

All data in Insights is stored in a PostgreSQL database rather than being read directly from the Perforce server. This has two advantages. Firstly, it means that when we run big queries we aren't impacting the performance of the Perforce server, and secondly, we can query the data using SQL which is a lot more expressive than what is possible with the Perforce DB files.

Though the format of the Perforce database is very good for doing what the server needs to do, it isn't so good for data collection queries.

Each table in the Perforce database has a matching table in the SQL database. Tables prefixed with `p4` represent the tables from the Perforce schema. Tables prefixed with `pa_` are internal to Insights.

You can see what's in your database by connecting to the PostgreSQL server and browsing the tables, columns and rows. You can also run your own SQL queries in the PostgreSQL client. Indeed, it is recommended that you try out queries in an SQL client before trying them out in a new plugin.

#### Interesting Tables

There are a lot of tables in Perforce, though not all are likely to be of interesting to you. The full Perforce schema can be found here:
However, some of the more interesting tables are given below. We only list tables and columns which are likely to be of interest in an Insights environment.

**File Revisions**

If you want to find files stored within Perforce, then the `p4rev` table is the place to look. This stores every revision that has existed, so many files will be listed multiple times. For example, if you wanted to find out how many files have a number of revisions, you could run:

```
select p4rev, count(*) from p4rev group by p4rev order by p4rev
```

The most commonly used columns in this table are as follows:

**Table 4.1. p4rev**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p4dfile</td>
<td>String</td>
<td>Depot path of file</td>
</tr>
<tr>
<td>p4rev</td>
<td>Integer</td>
<td>Revision number</td>
</tr>
<tr>
<td>p4change</td>
<td>Integer</td>
<td>The changelist that created the revision</td>
</tr>
<tr>
<td>p4date</td>
<td>Date</td>
<td>The date and time of the changelist</td>
</tr>
<tr>
<td>p4modtime</td>
<td>Date</td>
<td>The timestamp on the file when the revision was submitted</td>
</tr>
<tr>
<td>p4size</td>
<td>Long</td>
<td>The size of the file in bytes</td>
</tr>
</tbody>
</table>

**Submitted Changelists (p4change)**

This table contains information on all the changelists in the system. For example, to find which client workspaces have been responsible or the most submitted changelists, you could run:

```
select p4client, count(*) as total from p4change group by p4client order by total desc limit 10
```

**Table 4.2. p4change**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p4change</td>
<td>Integer</td>
<td>The change number</td>
</tr>
</tbody>
</table>
Chapter 4. Insights Plugin API

### Column 4.3. p4desc

<table>
<thead>
<tr>
<th>Column</th>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p4client</td>
<td>String</td>
<td>The client from which the change originates</td>
</tr>
<tr>
<td>p4user</td>
<td>String</td>
<td>The user who owns the change</td>
</tr>
<tr>
<td>p4date</td>
<td>Date</td>
<td>The date and time that the change was submitted</td>
</tr>
<tr>
<td>p4status</td>
<td>Integer</td>
<td>The status of the change (0=pending, 1=committed, 2=shelved, etc)</td>
</tr>
<tr>
<td>p4description</td>
<td>String</td>
<td>Short description of the change</td>
</tr>
<tr>
<td>p4root</td>
<td>String</td>
<td>Common path for all files in the changelist</td>
</tr>
</tbody>
</table>

If you want access to the full text description of a changelist, then these are stored in the `p4desc` table.

### Projects and Files

The `pa_project_files` table is not a native Perforce table, but is managed by Insights. If projects (together with their branches) have been configured in Insights, then this table is populated nightly with all the files in each project.

It references the `pa_project`, `pa_branches` and `pa_paths` tables.

If you wanted to find how many files are in each branch of all your projects, then you could do this with:

```sql
select p.name, b.name, count(f.file) from
    pa_project p, pa_project_files f, pa_branches b
where
    p.id = f.project_id and b.id = f.branch_id
group by p.name, b.name
order by p.name, b.name
```

### Table 4.3. pa_project_files

<table>
<thead>
<tr>
<th>Column</th>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>String</td>
<td>The full pathname of the file</td>
</tr>
<tr>
<td>project_id</td>
<td>Integer</td>
<td>The id of the project that the file belongs to</td>
</tr>
<tr>
<td>branch_id</td>
<td>Integer</td>
<td>The id of the branch that the file is in</td>
</tr>
</tbody>
</table>

### Jobs and Fixes

If you track jobs and bugs in Perforce, or use the DTG tool to import bugs in from an external bug tracking system (such as JIRA), then you can query information about bugs through the database.

`p4job` contains the list of all jobs, and `p4fix` relates jobs to changelists.
Table 4.4. p4job

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p4job</td>
<td>String</td>
<td>The job id (e.g. job012345)</td>
</tr>
<tr>
<td>p4desc</td>
<td>String</td>
<td>Short version of the job description</td>
</tr>
</tbody>
</table>

Table 4.5. p4fix

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p4job</td>
<td>String</td>
<td>The job id (e.g. job012345)</td>
</tr>
<tr>
<td>p4change</td>
<td>Integer</td>
<td>The changelist number</td>
</tr>
<tr>
<td>p4date</td>
<td>Date</td>
<td>The time and date the fix was recorded</td>
</tr>
<tr>
<td>p4status</td>
<td>String</td>
<td>The status of the job</td>
</tr>
<tr>
<td>p4client</td>
<td>String</td>
<td>The client where the fix originated</td>
</tr>
<tr>
<td>p4user</td>
<td>String</td>
<td>The user who fixed the job</td>
</tr>
</tbody>
</table>

There isn’t much of use in the p4job table, because all the interesting information is actually in p4bodtext. Unfortunately, because of how flexible jobs are, the entries in p4bodtext are effectively name-value pairs which can be difficult to query using SQL.

Table 4.6. p4bodtext

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p4key</td>
<td>String</td>
<td>Job id, link to the p4job table</td>
</tr>
<tr>
<td>p4attr</td>
<td>Integer</td>
<td>Id of the attribute for this job</td>
</tr>
<tr>
<td>p4text</td>
<td>String</td>
<td>Value for this attribute</td>
</tr>
</tbody>
</table>

The job attributes are defined in the job schema, which can be found by running `p4 jobspec -o`. This will list the job fields which have been defined, which for a default job specification will look like this:

```
101 Job word 32 required
102 Status select 10 required
103 User word 32 required
104 Date date 20 always
105 Description text 0 required
```

So to read the status and user of all bugs, you could run:
Chapter 4. Insights Plugin API

select j.p4job, b1.p4text as status, b2.p4text as user from  
    p4job j, p4bodtext b1, p4bodtext b2 where  
    j.p4job = b1.p4key and b1.p4attr = 102 and  
    j.p4job = b2.p4key and b2.p4attr = 103

Since job attribute 102 is the status, and 103 is the user who raised it.

The above is just a sample of the data that can be found in the Perforce schema. For a full explanation of what can be obtained from it, see the schema guide at http://ftp.perforce.com/perforce/r14.1/doc/schema/

Plugin to the Database

In the last section, we looked at the P4 schema and how that could be queried to pull data out of the Insights database. Before that, we looked at writing a simple plugin. This time, we’ll put both together to query the database and display that information in a chart.

Getting Data From the Database

The plugin API allows full access to the Insights database through the MetricQuery class. By calling this from the getData() method, it is possible to run full SQL queries and process the results that come back.

For example:

```groovy
MetricQuery query = new MetricQuery("SELECT p4dfile FROM p4rev LIMIT 10")

getSQL(query).each { row ->
    // Process the data
    println (row.p4dfile)
}
```

We can modify our HelloWorldMetric.groovy class from before to output real data by changing the getData() method to the following:
public DataWrapper getData() {
    DataWrapper wrapper = new DataWrapper()

    // Gets the title from the helloWorld.properties file.
    wrapper.title = getMessage("title")

    MetricQuery query = new MetricQuery("SELECT p4user, COUNT(*) AS num " +
    "FROM p4change GROUP BY p4user ORDER BY num DESC LIMIT ?")
    query.addInt(limit)
    def results = getSQL(query)

    def labels = [ ]
    def data = [ ]

    results.each { row ->
        labels.add(row.p4user)
        data.add(row.num)
    }

    // Example column chart.
    wrapper.ytitles = [ "0": "Number of submits" ]
    wrapper.xtitle = "Users"
    wrapper.xlabels = labels
    wrapper.data = data

    return wrapper
}

This runs a query against the p4change table, finding the users who have submitted the largest number of change lists. The query object can be parametrized, through the use of ? elements. You can then specify the actual value to use through the addX() methods, i.e.:

```
query.addInt()
query.addLong()
query.addString()
query.addDate()
```

Each of these must be called in the order that the ? are specified in the query. Alternatively, you could build up the SQL string by hand, but this may leave you open to SQL injection attacks and is not recommended.

The limit variable is automatically defined before getData() is called. It will default to 10 unless set by a filter parameter. Like all parameters that originate from the user, it is validated before being passed to getData().

Other variables that are passed are:

- **startDate** A date object (guaranteed to be before endDate).
- **endDate** A date object (guaranteed to be after startDate).
- **granularity** A Granularity enum, which can be DAYS, WEEKS, MONTHS, or RANGE.
limit An integer, greater than zero. This is the maximum number of rows to return for charts showing the 'Top N' results.

project If the PROJECTID filter is set, this will be set to a single `Project` object.

projects If the PROJECTS filter is set, this will be set to a list of zero or more `Project` objects.

users If the USERS filter is set, this will be set to a list of zero or more valid `User` objects.

Reading the data that comes back from the query is as simple as iterating over the results and referencing the result columns by name. In our example above, the `SELECT` query is pulling back `p4user, count(*) AS num`, so we can access `row.p4user` and `row.num` in each iteration.

What if we wanted to only see users in a particular project? As long as a project is being passed, then we could pull the users out of the `Project` object and add a `WHERE` clause which filtered on these. Unfortunately, we don’t have an easy way to setup a list of objects with `MetricQuery`, so we’d need to do some somewhat ugly code by hand:

```java
def where = "WHERE p4user IN ("
def field = "?"
    projects.get(0).users.each {
        where += field
        field = ",?"
    }
where += ")"

MetricQuery query = new MetricQuery("SELECT p4user, COUNT(*) AS num FROM +" + "p4change " + where + " GROUP BY p4user ORDER BY num DESC LIMIT ?")
    project.get(0).users.each {
        query.add(user.name)
    }
query.add(limit)
```

This assumes that users have been defined on the project, and at least one project was passed down to the plugin.

**Projects**

Each `Project` object has the following fields:

- **id** The unique project id
- **name** The name of the project
- **jobview** The job view
- **users** The list of users defined on the project

Projects are passed to the `getData()` method either as the `project` variable in in the `projects` list if this is a multi-project metric.

**Complex Queries**

Some queries that you might want to perform can be quite complicated and may take a long time to produce results. If this is the case, then you may want to consider running the queries overnight and
storing the results in your own table in a form that is quick to query. Some of the statistics that Insights uses are done this way. For example, the `pa_project_files` table is updated nightly.

If your plugin needs to perform intensive reporting tasks, implement a `dailyTask()` method. This method is called nightly in every plugin that implements it. You can run a query, perform calculations, create tables, and store your processed data in these tables. The plugin's `getData()` query can then run against these tables.

The `FileTypeMetric` that ships as an example with Insights shows this being done. This can be found in the same location as the `HelloWorld` plugin that was described earlier. The `getData()` method for this looks as follows:

```java
public DataWrapper getData() {
    DataWrapper     wrapper = new DataWrapper()
    wrapper.title = getMessage("fileTypes.title")

    MetricQuery query = new MetricQuery("SELECT filetype, SUM(total) AS total FROM xx_filetypes WHERE project_id = ? GROUP BY filetype ORDER BY total DESC LIMIT ?")
    query.addLong((long)project.id)
    query.addInt(limit)
    def results = getSQL(query)

    def labels = [ ]
    def data = [ ]

    results.each { row ->
        labels.add(row.filetype)
        data.add(row.total)
    }

    // Example column chart.
    wrapper.ytitles = [ "0": getMessage("fileTypes numberOfFiles") ]
    wrapper.xtitle = getMessage("fileTypes.types")
    wrapper.xlabels = labels
    wrapper.data = data

    return wrapper
}
```

This looks up files in the specified project (by reading `project.id` and passing it to `query.addLong()`), and reads data from the `xx_filetypes` table.

By convention, any table that has a name beginning with `xx_` is a custom extension table used by user-generated plugins, though there is no enforced restriction on the naming convention. This table is not normally created or updated by Insights, so how does it get there? This is where the `dailyTasks()` method on a plugin comes in.

Firstly, the plugin needs to inform Insights that it wants to run a daily task by implementing the `getTaskOrder()` method:
```java
public int getTaskOrder() {
    return 5
}
```

Plugins should be given an order between 1 and 10. Order 1 plugins are run before Order 10 plugins, and allow a plugin to depend on data written by another’s `dailyTask()`. If the order is less than 1, or greater than 10, the plugin’s daily task will not be executed.

The `dailyTask()` method itself is where all the work is actually done. For the `FileTypeMetric`, this is implemented as follows:
public void dailyTask() {
    try {
        if (!hasTable("xx_filetypes")) {
            updateSQL("CREATE TABLE xx_filetypes (project_id bigint, " +
                    "ext varchar(16), filetype varchar(16), total bigint")
        }
    } catch (Exception e) {
        println "Table already exists"
    }

    def i = 1
    def typesMap = [:]
    while (getMessageOrNull("types."+i+".label") != null) {
        String label = getMessageOrNull("types."+i+".label")
        String types = getMessageOrNull("types."+i+".types")
        typesMap.put(label, types.split(".", ""))
        i++
    }

    def projects = Project.findAll()
    // Cycle over each of the projects in turn.
    projects.each { project ->
        def map = [:]
        MetricQuery query =
            new MetricQuery("SELECT p.file as file FROM " +
                "pa_project_files p WHERE project_id=?")
        query.addLong(project.id)
        def results = getSQL(query)
        // For each file in the project...
        results.each() { row ->
            String file = row.file
            // Look for a dot which isn't the first character.
            if (file.indexOf("") > 0) {
                String ext = file.replaceAll(".*\.", ").replaceAll("[^a-zA-Z0-9]", ")
                if (ext.length() > 0) {
                    if (ext.length() > 16) {
                        ext = ext.substring(0, 16)
                    }
                    if (map.get(ext) == null) {
                        map.put(ext, 0)
                    }
                    map.put(ext, map.get(ext) + 1)
                }
            }
        }
        query = new MetricQuery("DELETE FROM xx_filetypes WHERE project_id=?")
        query.addLong(project.id)
        updateSQL(query)
        map.each() { ext, total ->
            String type = getFileType(typesMap, ext)
            query = new MetricQuery("INSERT INTO xx_filetypes (project_id, " +
                    "ext, filetype, total) VALUES(?, ?, ?, ?)")
            query.addLong(project.id)
            query.addString(ext)
            query.addString(type)
            query.addLong(total)
            updateSQL(query)
        }
    }
}
The first thing that the plugin must do here is ensure that the table it uses exists. The `hasTable()` method checks to see if the table exists, and if it doesn’t we create a new table. Note that plugins have the same access to the database as Insights does itself, so they have the freedom to create (or drop) any table they like.

Next up, `getMessageOrNull()` is used to read configuration from the properties file for the plugin. Since this is the `FileTypesMetric` plugin, this will be from the `fileTypes.properties` file.

Then a list of all projects known by Insights is retrieved, and a query is run to retrieve a list of each project’s files. This is a process that can take a long time, which is why it is being run as a daily task. Once we have the information we need, it is written into the `xx_filetypes` table by running an insert.

When the `getData()` method is called, we can read the pre-prepared data from the `xx_filetypes` table rather than having to recalculate it each time. Since each `dailyTask()` is run at night, any performance hit caused by running large queries should have less impact on the users.

Note that any plugin can read data from any table, so a single plugin could pre-generate data that is used by different metrics. For example, the `FileExtensions` metric also reads from the `xx_filetypes` table, but presents the results in a different way.

This short guide has hopefully given an overview of what is involved in extending Insights. The example plugins which are part of the distribution are another source of ideas which can be used when writing your plugin. For example, the `HelloWeather` plugin shows how to present data from sources other than the database.

## Classes

Insights provides the following API classes for plugin developers:

- “Chart” on page 37
- “DataWrapper” on page 41
- “Metric” on page 43
- “MetricQuery” on page 45

### Chart

The `Chart` class defines information about how a chart should be displayed. It should be defined and returned from the plugin’s `getChartOptions()` method. A simple use of the class is as follows:

```java
Chart chart = new Chart()
chart.seriesGroups.put("root", [ "type": "line", "values": "number" ])
chart.options.put("basic", [ "stacked": false, "fill": true, "legend": false, "zoom": false ])
chart.options.put("thumbnail", [ "cursor": false ])
chart.options.put("expanded", [ "cursor": true ])
chart.dataType = Chart.DataType.RANGE
chart.filters.add(Chart.Filter.PROJECTS)
chart.filters.add(Chart.Filter.DATERANGE)
```

If the `seriesGroups` key is "root" then the chart data will be rendered as a single set of values.
If the `seriesGroups` key is "0" then the chart data will support stacked series data.

The series group takes the parameters:

- **type** - can be **line** or **column**
- **values** - can be **number** or **percent**

**Options**

Options must be set for the three keys **basic**, **thumbnail** and **expanded**. Options for **basic** are inherited by the other two.

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>stacked</td>
<td>If <strong>true</strong>, multiple series are stacked on top of each other, otherwise they are shown next to each other.</td>
</tr>
<tr>
<td>fill</td>
<td>If <strong>true</strong>, the graph is filled otherwise it is left empty.</td>
</tr>
<tr>
<td>label</td>
<td>If <strong>true</strong>, display a value label within the chart.</td>
</tr>
<tr>
<td>legend</td>
<td>If <strong>true</strong>, display a legend below the chart.</td>
</tr>
<tr>
<td>zoom</td>
<td>If <strong>true</strong>, allow user to zoom into the chart.</td>
</tr>
<tr>
<td>cursor</td>
<td>If <strong>true</strong>, on hover display balloons for all values in stack, otherwise only show balloons on one value.</td>
</tr>
<tr>
<td>animateupto</td>
<td>A numeric value. If there are more values than this, then do not perform animation when displaying the chart. This can be used to speed up render time for a graph.</td>
</tr>
</tbody>
</table>

**Enums**

**DataType**

Defines the type of chart data that is being displayed, and is used to set the `dataType` field on the `Chart` object. If set to **PERIOD**, then the data is expected to consist of a number of values each for a different time period, such as the number of submits for each day over the last month. Each day will be displayed as it's own column.

For **RANGE** data, only one total value is shown over the entire period, so the total number of submits for the last month. **RANGE** data is often shown for 'Top N' charts, for example showing the users who have made the most submits for the last month.

**LIVE** data is being updated continuously. This type of chart is not currently supported by plugins, but is used by some internal metrics.

- **PERIOD**
- **RANGE**
Filter

Filters can be selected by users to restrict the results displayed in the chart. For example, a metric that displays information about a project will have a PROJECTID filter to allow the user to select which project to display. A metric will only display filters for those added to the Chart object. Which filters are available also control what type of dashboard a metric will appear on. A metric with a PROJECTID filter will be displayed on the “Single Project” dashboard, whilst one with the PROJECTS filter will be displayed on the "Multi Project" dashboard.

- GRANULARITY
- DATERANGE
- PROJECTID
- PROJECTS
- USERS
- JOBSTATES

Fields

Each of the fields is set to an empty list or map when the class is constructed:

- Map<String, Map<String, String>> seriesGroups - Definitions of series groups to define the type of chart.
- Map<String, Map<String, Boolean>> options - Toggle options to control how the charts are displayed.
- List<Filter> filters - Defines which filters are used for this chart. See the definition of “Filter” on page 39.
- DataTypes dataType - The type of data being displayed by the chart. See the definition of “DataType” on page 38.
**DataWrapper**

Wraps data for metric results. Contains graph/table labelling information which is passed to the client. Also contains all the data that is to be displayed. Data consists of one or more series. Each series may be stacked, or shown as multiple columns under a single label. Each series has a label defined for it. Data for each series is a flat array, which is sorted according to the labels provided. All label and data arrays will be the same length and have the same sort order.

A DataWrapper object is returned from the `Metrics.getData()` method, so a plugin must instantiate this object and set the fields to include the chart data. A simple example of this is as follows:

```java
DataWrapper wrapper = new DataWrapper()
  wrapper.title   = "Simple Chart"
  wrapper.ytitles = [ "0": "Number" ]
  wrapper.xtitle  = "Users"
  wrapper.xlabels = [ "Alice", "Bob", "Eve" ]
  wrapper.data    = [ 17, 13, 23 ]
```

**Fields**

The following fields can be set by the user:

- **String title** - Title for the graph.
- **String xtitle** - Title for the x-axis of the graph.
- **Map<String, String> ytitles** - Map of titles for the y-axes of the graph.
- **List<String> xlabels** - Labels for each column in the x-axis.
- **List<DataSeries> series** - Series data. Will be one or more.
- **List<integer> data** - Non-series data.

Either the `series` or `data` fields should be set, but not both; exactly one of them must be `null`.
Chapter 4. Insights Plugin API

Metric

This is the super class of all metric plugins. It provides fields for all the filters, plus service definitions for access to helper methods within the core application. To implement a filter, at a minimum `getChartOptions()` and `getData()` need to be completed. The other methods are helper methods should they be needed. When the user requests data from a metric, the `getData()` method is called. At point of call, all the fields will have been set to sanitized values.

User request fields

These fields are set whenever there is a user request for data from the metric, i.e. just before `getData()` is called. By the time `getData()` is called, they will have been validated and set to suitable default values if absent.

- **String** `metricName` - The name of the metric. Always set.
- **Date** `startDate` - The start date of the query to display data from. Always set.
- **Date** `endDate` - The end date of the query to display data to. Always set.
- **Granularity** `granularity` - The granularity date dependant data should be returned in. Always set.
- **Project** `project` - For a single project widget, the project that was requested.
- **Set** `<Project>` `projects` - For a multi project widget, the set of projects to filter on.
- **Set** `<User>` `users` - The set of users to filter on.
- **Set** `<String>` `states` - The set of job states to filter on.
- **int** `limit` - The maximum number of results to return. Only used by the 'top N' metrics. Always set.
- **Map** `<String, Object>` `params` - List of HTTP parameters that were unrecognized. This allows a metric to support its own unique parameters, but these will not be validated prior to `getData()` being called.

Internal fields

A plugin also has access to the following global objects:

- **HttpSession session** - Provides direct access to the HTTP session object.
- **GrailsApplication grailsApplication** - Provides direct access to the Grails application object.

Methods

`dailyTask()`

To be optionally implemented by a plugin. Run every night as part of the daily statistics generation after all other generation has been completed. If generation of historical data is supported, then this method should ideally just call `dailyTask(new Date().clearTime(), null)`, to avoid duplicating functionality.
**dailyTask(Date startDate, Date endDate)**

To be optionally implemented by a plugin. This is run when generating historical data. If a metric supports historical data, then this method should be implemented. Such metrics will probably implement `dailyTask()` to simply call this method with today's date. The date to generate statistics for is passed in. This will be the date with a time of midnight (00:00:00). If the `endDate` is non-null, then data should be generated for every day in the range (inclusive).

Parameters:

- **Date startDate** - to generate historical data for.
- **Date endDate** - to end generating data for.

**Chart getChartOptions()**

Gets the current set of chart options for widgets using this metric. These options describe how the metric data should be presented within the Insights UI. This needs to be implemented by a plugin.

**DataWrapper getData()**

To be implemented by a plugin. All field variables are guaranteed to have been set to sanitized values by the time that this is called.

**String getMessage(String code)**

Gets the localized message from the message file for this metric. If the message cannot be found, an error message is returned that is suitable for being returned to the user.

**String getMessageOrNull(string code)**

 Gets the localized message from the message file for this metric. If the message cannot be found, a null string is returned.

**List getSQL(MetricQuery query)**

Executes an SQL query against the database and returns the result set. This is the preferred version of `getSQL()` since it uses a safe parametrized object rather than a `String`. Results are returned as a list, with each row an object with fields named for the table row.

**void updateSQL(MetricQuery query)**

Executes a parametrized SQL update against the database. This is the preferred version of `updateSQL()` since it is safe against SQL injection attacks.

**int getTaskOrder()**

Gets the order of metric tasks to be run nightly. This should be a number from 1 to 10. Tasks with order 1 run first, and tasks with order 10 run last. Tasks with orders outside this range are never run. The sequence of processing tasks with the same order is not guaranteed. Unless it is important that this task is run before another, 5 is recommended.

**boolean hasTable(String tableName)**

Returns true if the named table exists in the database.
**MetricQuery**

This is a wrapper to an SQL statement. It allows parameters to be set so that they can be passed in a safe manner. Parameters are always read in the order that they are set. In order to use it, do something like the following:

```java
MetricQuery query = new MetricQuery("SELECT * FROM foo WHERE a > ? AND a < ?")
query.addInt(1)
query.addInt(10)
```

Parameters must be set in the order that the `?` operators appear in the SQL string.

**Constructors**

**MetricQuery(String query)**

The base constructor defines and sets an SQL statement. Use `?` to specify placeholders for parameters. Parameters must then be added in the order they are specified in the SQL string using the `add*()` methods.

**Methods**

**void addDate(Date value)**

Adds a date parameter as the next value in the list.

**void addInt(int value)**

Adds an int parameter as the next value in the list.

**void addLong(long value)**

Adds a long parameter as the next value in the list.

**void addString(String value)**

Adds a String parameter as the next value in the list.

**List getParams()**

Gets the list of all the parameters, in the order that they were set. Each parameter consists of an object in the format [ *type: x, value: y* ]. The type is of class `java.sql.Type`, the value is an `Object` of a suitable type.

**String getQuery()**

Gets the query string, without any parameter substitution.
Data types

Granularity (enum)

Defines and provides useful functions to manage the splitting of time periods. Metrics will return data at a certain level of time granularity. For example, one entry per day, or one entry per month. Given a random date, this enum can be used to agglomerate dates which fall into the same period, and sort them in a consistent fashion. Granularities currently supported are:

- DAYS
- WEEKS
- MONTHS
- RANGE

The RANGE value is special, in that it defines a single entry that covers the entire range.

The format() method can be used to generate keys when agglomerating data, as the following example shows:

```java
results = getSQL(query)
results.each() { row ->
    Date date = Granularity.MONTH.format(row.date)
    if (map[date] == null) {
        map[date] = 0
    }
    map[date] = map[date] + row.value
}
```

You can also use the next() method to iterate over a complete date range to make sure that you have no gaps in the range.

Methods

String format(Date date)

Gets the formatted date suitable for this level of granularity. This will use formats of yyyy/MM/dd, yyyy/ww and yyyy/MM for DAYS, WEEKS and MONTHS respectively. RANGE dates are always returned as a constant string. This allows sorting and indexing of dates in a consistently unique way.

Date floor(Date date)

Gets a date to the start of its current period. For a year granularity, the date is set to be the 1st of January, for a month the 1st of that month, for a day the start of the date.

Date ceiling(Date date)

Gets a date at the end of the current period. This will be the last day of the month or year, depending on the granularity.
Date next(Date date)

Gets the next date after the given date. This will skip forward a day, week, or month, depending on the granularity.
License Statements

Perforce software includes software developed by the University of California, Berkeley and its contributors. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

Perforce software includes software from the Apache ZooKeeper project, developed by the Apache Software Foundation and its contributors. (http://zookeeper.apache.org/)

For the complete list of third-party licenses, click your user name on the Insights toolbar and select About Insights.