pgmoneta

Developer Guide

Contents

1 Introduction

pgmoneta is a backup / restore solution for PostgreSQL.

Ideally, you would not need to do backups and disaster recovery, but that isn't how the real World works.

Possible scenarios that could happen

- Data corruption
- System failure
- Human error
- Natural disaster

and then it is up to the database administrator to get the database system back on-line, and to the correct recovery point.

Two key factors are

- Recovery Point Objective (RPO): Maximum targeted period in which data might be lost from an IT service due to a major incident
- Recovery Time Objective (RTO): The targeted duration of time and a service level within which a business process must be restored after a disaster (or disruption) in order to avoid unacceptable consequences associated with a break in business continuity

You would like to have both of these as close to zero as possible, since RPO of 0 means that you won't lose data, and RTO of 0 means that your system recovers at once. However, that is easier said than done.

pgmoneta is focused on having features that will allow database systems to get as close to these goals as possible such that high availability of 99.99% or more can be implemented, and monitored through standard tools.

pgmoneta is named after the Roman Goddess of Memory.

1.1 Features

- Full backup
- Restore
- Compression (gzip, zstd, lz4, bzip2)
- AES encryption support
- Symlink support
- WAL shipping support

- Hot standby
- Prometheus support
- Remote management
- Offline detection
- Transport Layer Security (TLS) v1.2+ support
- Daemon mode
- User vault

1.2 Platforms

The supported platforms are

- Fedora 38+
- RHEL 9
- RockyLinux 9
- FreeBSD
- OpenBSD

2 Installation

2.1 Fedora

You need to add the PostgreSQL YUM repository, for example for Fedora 40

```
dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/F
   -40-x86_64/pgdg-fedora-repo-latest.noarch.rpm
```

and do the install via

```
dnf install -y pgmoneta
```

Additional information

- PostgreSQL YUM
- Linux downloads

2.2 RHEL 9 / RockyLinux 9

x86_64

```
dnf install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest
   -9.noarch.rpm
dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL
   -9-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

aarch64

```
dnf install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest
   -9.noarch.rpm
dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL
   -9-aarch64/pgdg-redhat-repo-latest.noarch.rpm
```

and do the install via

```
dnf install -y pgmoneta
```

2.3 Compiling the source

We recommend using Fedora to test and run **pgmoneta**, but other Linux systems, FreeBSD and MacOS are also supported.

pgmoneta requires

- clang
- cmake
- make
- libev
- OpenSSL
- zlib
- zstd
- lz4
- bzip2
- systemd
- rst2man
- libssh
- libarchive

```
dnf install git gcc clang clang-analyzer cmake make libev libev-devel \
            openssl openssl-devel \
            systemd systemd-devel zlib zlib-devel \
            libzstd libzstd-devel \
            lz4 lz4-devel libssh libssh-devel \
            python3-docutils libatomic \
            bzip2 bzip2-devel \
            libarchive libarchive-devel
```

Alternative gcc can be used.

2.3.1 RHEL / RockyLinux

On RHEL / Rocky, before you install the required packages some additional repositories need to be enabled or installed first.

First you need to install the subscription-manager

```
dnf install subscription-manager
```

It is ok to disregard the registration and subscription warning.

Otherwise, if you have a Red Hat corporate account (you need to specify the company/organization name in your account), you can register using

```
subscription-manager register --username <your-account-email-or-login> --
   password <your-password> --auto-attach
```

Then install the EPEL repository,

```
dnf install epel-release
```

Then to enable powertools

```
dnf config-manager --set-enabled codeready-builder-for-rhel-9-rhui-rpms
dnf config-manager --set-enabled crb
dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.
   noarch.rpm
```

Then use the `dnf` command for **pgmoneta** to install the required packages.

2.3.2 FreeBSD

On FreeBSD, `pkg` is used instead of `dnf` or `yum`.

Use `pkg install <package name>` to install the following packages

```
git gcc cmake libev openssl libssh zlib-ng zstd liblz4 bzip2 \
    py39-docutils libarchive
```

2.3.3 Build

2.3.3.1 Release build

The following commands will install **pgmoneta** in the `/usr/local` hierarchy.

```
git clone https://github.com/pgmoneta/pgmoneta.git
cd pgmoneta
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install
```

See RPM for how to build a RPM of **pgmoneta**.

2.3.3.2 Debug build

The following commands will create a `DEBUG` version of **pgmoneta**.

```
git clone https://github.com/pgmoneta/pgmoneta.git
cd pgmoneta
mkdir build
cd build
cmake -DCMAKE_C_COMPILER=clang -DCMAKE_BUILD_TYPE=Debug ..
make
```

You can do

```
cmake -DCMAKE_C_COMPILER=clang -DCMAKE_BUILD_TYPE=Debug -DCMAKE_C_FLAGS="-DCORE_DEBUG" ..
```

in order to get information from the core libraries too.

2.4 Compiling the documentation

pgmoneta’s documentation requires

- pandoc
- texlive

```
dnf install pandoc texlive-scheme-basic \
            ’tex(footnote.sty)’ ’tex(footnotebackref.sty)’ \
            ’tex(pagecolor.sty)’ ’tex(hardwrap.sty)’ \
            ’tex(mdframed.sty)’ ’tex(sourcesanspro.sty)’ \
            ’tex(ly1enc.def)’ ’tex(sourcecodepro.sty)’ \
            ’tex(titling.sty)’ ’tex(csquotes.sty)’ \
            ’tex(zref-abspage.sty)’ ’tex(needspace.sty)’ \
            ’tex(selnolig.sty)’
```

You will need the `Eisvogel` template as well which you can install through

```
wget https://github.com/Wandmalfarbe/pandoc-latex-template/releases/download/v3.2.0/Eisvogel-3.2.0.tar.gz
tar -xzf Eisvogel-3.2.0.tar.gz
mkdir -p $HOME/.local/share/pandoc/templates
mv Eisvogel-3.2.0/eisvogel.latex $HOME/.local/share/pandoc/templates
```

where `$HOME` is your home directory.

2.4.0.1 Generate API guide

This process is optional. If you choose not to generate the API HTML files, you can opt out of downloading these dependencies, and the process will automatically skip the generation.

Download dependencies

```
dnf install graphviz doxygen
```

2.4.1 Build

These packages will be detected during `cmake` and built as part of the main build.

2.5 Extension installation

When you configure the extra parameter in the server section of pgmoneta.conf, it requires the server side to have the pgmoneta_ext extension installed to make it work.

The following instructions can help you easily install pgmoneta_ext. If you encounter any problems, please refer to the more detailed instructions in the DEVELOPERS documentation.

2.5.1 Install pgmoneta_ext

After you have successfully installed pgmoneta, the following commands will help you install pgmoneta_ext:

```
dnf install -y pgmoneta_ext
```

You need to add the pgmoneta_ext library for PostgreSQL in postgrersql.conf as well:

```
shared_preload_libraries = 'pgmoneta_ext'
```

And remember to restart PostgreSQL to make it work.

2.5.2 Verify success

You can use the postgres role to test.

1. Log into PostgreSQL

   ```
   psql
   ```

2. Create a new test database

   ```
   CREATE DATABASE testdb;
   ```

3. Enter the database

   ```
   \c testdb
   ```

4. Follow the SQL commands below to check the function

   ```
   DROP EXTENSION IF EXISTS pgmoneta_ext;
   CREATE EXTENSION pgmoneta_ext;
   SELECT pgmoneta_ext_version();
   ```

You should see

```
 pgmoneta_ext_version
----------------------
 0.1.0
(1 row)
```

2.5.3 Granting SUPERUSER Privileges

Some functions in `pgmoneta_ext` require `SUPERUSER` privileges. To enable these, grant the `repl` role superuser privileges using the command below. **Please proceed with caution**: granting superuser privileges bypasses all permission checks, allowing unrestricted access to the database, which can pose security risks. We are committed to enhancing privilege security in future updates.

```
ALTER ROLE repl WITH SUPERUSER;
```

To revoke superuser privileges from the `repl` role, use the following command:

```
ALTER ROLE repl WITH NOSUPERUSER;
```

3 C programming

pgmoneta is developed using the C programming language so it is a good idea to have some knowledge about the language before you begin to make changes.

There are books like,

- C in a Nutshell
- 21st Century C

that can help you

3.1 Debugging

In order to debug problems in your code you can use gdb, or add extra logging using the `pgmoneta_log_XYZ()` API

4 Git guide

Here are some links that will help you

- How to Squash Commits in Git
- ProGit book

4.1 Basic steps

4.1.1 Start by forking the repository

This is done by the "Fork" button on GitHub.

4.2 Clone your repository locally

This is done by

```
git clone git@github.com:<username>/pgmoneta.git
```

4.2.1 Add upstream

Do

```
cd pgmoneta
git remote add upstream https://github.com/pgmoneta/pgmoneta.git
```

4.2.2 Do a work branch

```
git checkout -b mywork main
```

4.2.3 Make the changes

Remember to verify the compile and execution of the code.

Use

```
[#xyz] Description
```

as the commit message where `[#xyz]` is the issue number for the work, and `Description` is a short description of the issue in the first line

4.2.4 Multiple commits

If you have multiple commits on your branch then squash them

```
git rebase -i HEAD~2
```

for example. It is `p` for the first one, then `s` for the rest

4.2.5 Rebase

Always rebase

```
git fetch upstream
git rebase -i upstream/main
```

4.2.6 Force push

When you are done with your changes force push your branch

```
git push -f origin mywork
```

and then create a pull request for it

4.2.7 Format source code

Use

```
./uncrustify.sh
```

to format the source code

4.2.8 Repeat

Based on feedback keep making changes, squashing, rebasing and force pushing

4.2.9 Undo

Normally you can reset to an earlier commit using `git reset <commit hash> --hard`.

But if you accidentally squashed two or more commits, and you want to undo that, you need to know where to reset to, and the commit seems to have lost after you rebased.

But they are not actually lost - using `git reflog`, you can find every commit the HEAD pointer has ever pointed to. Find the commit you want to reset to, and do `git reset --hard`.

5 Architecture

5.1 Overview

pgmoneta use a process model (`fork()`), where each process handles one Write-Ahead Log (WAL) receiver to PostgreSQL.

The main process is defined in main.c.

Backup is handled in backup.h (backup.c).

Restore is handled in restore.h (restore.c) with linking handled in link.h (link.c).

Archive is handled in achv.h (archive.c) backed by restore.

Write-Ahead Log is handled in wal.h (wal.c).

Backup information is handled in info.h (info.c).

Retention is handled in retention.h (retention.c).

Compression is handled in gzip_compression.h (gzip_compression.c), lz4_compression.h (lz4_compression.c), zstandard_compression.h (zstandard_compression.c), and bzip2_compression.h (bzip2_compression.c).

Encryption is handled in aes.h (aes.c).

5.2 Shared memory

A memory segment (shmem.h) is shared among all processes which contains the **pgmoneta** state containing the configuration and the list of servers.

The configuration of **pgmoneta** (`struct configuration`) and the configuration of the servers (`struct server`) is initialized in this shared memory segment. These structs are all defined in pgmoneta.h.

The shared memory segment is created using the `mmap()` call.

5.3 Network and messages

All communication is abstracted using the `struct message` data type defined in messge.h.

Reading and writing messages are handled in the message.h (message.c) files.

Network operations are defined in network.h (network.c).

5.4 Memory

Each process uses a fixed memory block for its network communication, which is allocated upon startup of the process.

That way we don't have to allocate memory for each network message, and more importantly free it after end of use.

The memory interface is defined in memory.h (memory.c).

5.5 Management

pgmoneta has a management interface which defines the administrator abilities that can be performed when it is running. This include for example taking a backup. The `pgmoneta-cli` program is used for these operations (cli.c).

The management interface is defined in management.h. The management interface uses its own protocol which uses JSON as its foundation.

5.5.1 Write

The client sends a single JSON string to the server,

Field	Type	Description
`compression`	uint8	The compression type
`encryption`	uint8	The encryption type
`length`	uint32	The length of the JSON document
`json`	String	The JSON document

The server sends a single JSON string to the client,

Field	Type	Description
`compression`	uint8	The compression type
`encryption`	uint8	The encryption type
`length`	uint32	The length of the JSON document

Field	Type	Description
`json`	String	The JSON document

5.5.2 Read

The server sends a single JSON string to the client,

Field	Type	Description
`compression`	uint8	The compression type
`encryption`	uint8	The encryption type
`length`	uint32	The length of the JSON document
`json`	String	The JSON document

The client sends to the server a single JSON documents,

Field	Type	Description
`compression`	uint8	The compression type
`encryption`	uint8	The encryption type
`length`	uint32	The length of the JSON document
`json`	String	The JSON document

5.5.3 Remote management

The remote management functionality uses the same protocol as the standard management method.

However, before the management packet is sent the client has to authenticate using SCRAM-SHA-256 using the same message format that PostgreSQL uses, e.g. StartupMessage, AuthenticationSASL, AuthenticationSASLContinue, AuthenticationSASLFinal and AuthenticationOk. The SSLRequest message is supported.

The remote management interface is defined in remote.h (remote.c).

5.6 libev usage

libev is used to handle network interactions, which is "activated" upon an `EV_READ` event.

Each process has its own event loop, such that the process only gets notified when data related only to that process is ready. The main loop handles the system wide "services" such as idle timeout checks and so on.

5.7 Signals

The main process of **pgmoneta** supports the following signals `SIGTERM`, `SIGINT` and `SIGALRM` as a mechanism for shutting down. The `SIGABRT` is used to request a core dump (`abort()`).

The `SIGHUP` signal will trigger a reload of the configuration.

It should not be needed to use `SIGKILL` for **pgmoneta**. Please, consider using `SIGABRT` instead, and share the core dump and debug logs with the **pgmoneta** community.

5.8 Reload

The `SIGHUP` signal will trigger a reload of the configuration.

However, some configuration settings requires a full restart of **pgmoneta** in order to take effect. These are

- `hugepage`
- `libev`
- `log_path`
- `log_type`
- `unix_socket_dir`
- `pidfile`

The configuration can also be reloaded using `pgmoneta-cli -c pgmoneta.conf conf reload`. The command is only supported over the local interface, and hence doesn't work remotely.

The `SIGHUP` signal will trigger a full reload of the configuration. When `SIGHUP` is received, **pgmoneta** will re-read the configuration from the configuration files on disk and apply any changes that can be handled at runtime. This is the standard way to apply changes made to the configuration files.

In contrast, the `SIGUSR1` signal will trigger a service reload, but **does not** re-read the configuration files. Instead, `SIGUSR1` restarts specific services (metrics and management) using the current in-memory configuration. This signal is automatically triggered by `pgmoneta-cli conf set` when applying runtime configuration changes that don't require a full restart. Any changes made to the configuration files will **not** be picked up when using `SIGUSR1`; only the configuration already loaded in memory will be used.

Services affected by SIGUSR1: - **Metrics service**: Restarted when `metrics` port is changed or enabled/disabled - **Management service**: Restarted when `management` port is changed or enabled/disabled

5.9 Prometheus

pgmoneta has support for Prometheus when the `metrics` port is specified.

The module serves two endpoints

- / - Overview of the functionality (`text/html`)
- /`metrics` - The metrics (`text/plain`)

All other URLs will result in a 403 response.

The metrics endpoint supports `Transfer-Encoding: chunked` to account for a large amount of data.

The implementation is done in prometheus.h and prometheus.c.

5.10 Logging

Simple logging implementation based on a `atomic_schar` lock.

The implementation is done in logging.h and logging.c.

5.11 Protocol

The protocol interactions can be debugged using Wireshark or pgprtdbg.

6 Encryption

6.1 Overview

AES Cipher block chaining (CBC) mode and AES Counter (CTR) mode are supported in **pgmoneta**. The default setup is no encryption.

CBC is the most commonly used and considered save mode. Its main drawbacks are that encryption is sequential (decryption can be parallelized).

Along with CBC, CTR mode is one of two block cipher modes recommended by Niels Ferguson and Bruce Schneier. Both encryption and decryption are parallelizable.

Longer the key length, safer the encryption. However, with 20% (192 bit) and 40% (256 bit) extra workload compare to 128 bit.

6.2 Encryption Configuration

`none`: No encryption (default value)

`aes | aes-256 | aes-256-cbc`: AES CBC (Cipher Block Chaining) mode with 256 bit key length

`aes-192 | aes-192-cbc`: AES CBC mode with 192 bit key length

`aes-128 | aes-128-cbc`: AES CBC mode with 128 bit key length

`aes-256-ctr`: AES CTR (Counter) mode with 256 bit key length

`aes-192-ctr`: AES CTR mode with 192 bit key length

`aes-128-ctr`: AES CTR mode with 128 bit key length

6.3 Encryption / Decryption CLI Commands

6.3.1 decrypt

Decrypt the file in place, remove encrypted file after successful decryption.

Command

```
pgmoneta-cli decrypt <file>
```

6.3.2 encrypt

Encrypt the file in place, remove unencrypted file after successful encryption.

Command

```
pgmoneta-cli encrypt <file>
```

6.4 Benchmark

Check if your CPU have AES-NI

```
cat /proc/cpuinfo | grep aes
```

Query number of cores on your CPU

```
lscpu | grep '^CPU(s):'
```

By default openssl using AES-NI if the CPU have it.

```
openssl speed -elapsed -evp aes-128-cbc
```

Speed test with explicit disabled AES-NI feature

```
OPENSSL_ia32cap="~0x200000200000000" openssl speed -elapsed -evp aes-128-
    cbc
```

Test decrypt

```
openssl speed -elapsed -decrypt -evp aes-128-cbc
```

Speed test with 8 cores

```
openssl speed -multi 8 -elapsed -evp aes-128-cbc
```

```
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
    CPU family:          6
    Model:               158
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            10
    BogoMIPS:            5183.98
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
       pge mca cmov pat pse36 clflush mmx fxsr sse sse2 s
                         s ht syscall nx pdpe1gb rdtscp lm constant_tsc
                            rep_good nopl xtopology cpuid pni pclmulqdq
                            vmx ssse
                         3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes
                            xsave avx f16c rdrand hypervisor lahf_lm abm 3
                            dnowpr
                         efetch invpcid_single pti ssbd ibrs ibpb stibp
                            tpr_shadow vnmi ept vpid ept_ad fsgsbase bmi1
                            avx2 s
                         mep bmi2 erms invpcid rdseed adx smap clflushopt
                            xsaveopt xsavec xgetbv1 xsaves flush_l1d
                            arch_capa
                         bilities
Virtualization features:
  Virtualization:        VT-x
  Hypervisor vendor:     Microsoft
  Virtualization type:   full
Caches (sum of all):
  L1d:                   192 KiB (6 instances)
  L1i:                   192 KiB (6 instances)
  L2:                    1.5 MiB (6 instances)
  L3:                    12 MiB (1 instance)
Vulnerabilities:
  Itlb multihit:         KVM: Mitigation: VMX disabled
  L1tf:                  Mitigation; PTE Inversion; VMX conditional cache
     flushes, SMT vulnerable
  Mds:                   Vulnerable: Clear CPU buffers attempted, no
     microcode; SMT Host state unknown
  Meltdown:              Mitigation; PTI
  Spec store bypass:     Mitigation; Speculative Store Bypass disabled via
      prctl and seccomp
  Spectre v1:            Mitigation; usercopy/swapgs barriers and __user
     pointer sanitization
  Spectre v2:            Mitigation; Full generic retpoline, IBPB
     conditional, IBRS_FW, STIBP conditional, RSB filling
  Srbds:                 Unknown: Dependent on hypervisor status
  Tsx async abort:       Not affected

openssl version: 3.0.5
built on: Tue Jul  5 00:00:00 2022 UTC
options: bn(64,64)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -O2 -flto=auto -ffat-
   lto-objects -fexceptions -g -grecord-gcc-switches -pipe -Wall -Werror=
   format-security -Wp,-D_FORTIFY_SOURCE=2 -Wp,-D_GLIBCXX_ASSERTIONS -
   specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -fstack-protector-strong
   -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1  -m64  -mtune=generic -
   fasynchronous-unwind-tables -fstack-clash-protection -fcf-protection -
```

```
    O2 -flto=auto -ffat-lto-objects -fexceptions -g -grecord-gcc-switches -
    pipe -Wall -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -Wp,-
    D_GLIBCXX_ASSERTIONS -specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -
    fstack-protector-strong -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -
    m64 -mtune=generic -fasynchronous-unwind-tables -fstack-clash-
    protection -fcf-protection -Wa,--noexecstack -Wa,--generate-missing-
    build-notes=yes -specs=/usr/lib/rpm/redhat/redhat-hardened-ld -specs=/
    usr/lib/rpm/redhat/redhat-annobin-cc1 -DOPENSSL_USE_NODELETE -DL_ENDIAN
     -DOPENSSL_PIC -DOPENSSL_BUILDING_OPENSSL -DZLIB -DNDEBUG -DPURIFY -
    DDEVRANDOM="\"/dev/urandom\"" -DSYSTEM_CIPHERS_FILE="/etc/crypto-
    policies/back-ends/openssl.config"
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192
    bytes  16384 bytes
AES-128-CBC *   357381.06k   414960.06k   416301.23k   416687.10k
    416175.45k   416268.29k
AES-128-CBC     902160.83k  1496344.68k  1514778.62k  1555236.52k
    1542537.22k  1569259.52k
AES-128-CBC d   909710.79k  2941259.46k  5167110.31k  5927086.76k
    6365967.70k  6349198.68k
AES-128-CBC 8  3912786.36k  8042348.31k  9870507.86k 10254096.38k
    10653332.82k 10310331.05k
AES-128-CBC 8d 4157037.26k 12337480.36k 26613686.27k 29902703.27k
    32306793.13k 31440366.25k

AES-128-CTR *   146971.83k   165696.94k   574871.64k   634507.61k
    676448.94k   668139.52k
AES-128-CTR     887783.06k  2255074.22k  4800168.19k  5930596.01k
    6431110.49k  6376062.98k
AES-128-CTR d   793432.63k  2181439.06k  4541298.09k  5743022.42k
    6480090.45k  6271221.76k
AES-128-CTR 8  3833975.47k 10832239.55k 23757293.40k 28413146.79k
    30514317.99k 30092356.27k
AES-128-CTR 8d 3456838.44k  9749773.91k 22107652.18k 27229352.28k
    30703026.18k 29387025.07k

AES-192-CBC     853380.50k  1238507.90k  1299788.12k  1257189.03k
    1272591.70k  1271840.77k
AES-192-CBC d   876094.29k  2843770.82k  4523019.52k  5177496.92k
    5442652.84k  5372559.36k
AES-192-CTR     869039.84k  2285946.18k  4229439.91k  5049118.04k
    5422994.77k  5309748.57k
AES-192-CTR d   789470.51k  2177050.05k  4194812.76k  4935891.63k
    5257865.90k  5323046.91k

AES-256-CBC     834298.24k  1100648.64k  1117826.90k  1104301.40k
    1130657.11k  1097285.63k
AES-256-CBC d   843079.68k  2714917.67k  4084088.23k  4510005.59k
    4557821.27k  4594783.57k
AES-256-CTR     811325.74k  2222582.89k  3749333.08k  4412143.27k
    4640549.55k  4554828.46k
```

```
AES-256-CTR d   730844.97k  2081179.20k  3673258.15k  4346793.64k
    4515722.58k  4594335.74k
```

*: AES-NI disabled; 8: 8 cores; d: decryption

7 RPM

pgmoneta can be built into a RPM for Fedora systems.

7.1 Requirements

```
dnf install gcc rpm-build rpm-devel rpmlint make python bash coreutils
    diffutils patch rpmdevtools chrpath
```

7.2 Setup RPM development

```
rpmdev-setuptree
```

7.3 Create source package

```
git clone https://github.com/pgmoneta/pgmoneta.git
cd pgmoneta
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make package_source
```

7.4 Create RPM package

```
cp pgmoneta-$VERSION.tar.gz ~/rpmbuild/SOURCES
QA_RPATHS=0x0001 rpmbuild -bb pgmoneta.spec
```

The resulting RPM will be located in `~/rpmbuild/RPMS/x86_64/`, if your architecture is `x86_64`
.

8 Building pgmoneta

8.1 Overview

pgmoneta can be built using CMake, where the build system will detect the compiler and apply appropriate flags for debugging and testing.

The main build system is defined in [CMakeLists.txt][cmake_txt]. The flags for Sanitizers are added in compile options in [src/CMakeLists.txt][src/cmake_txt]

8.2 Dependencies

Before building pgmoneta with sanitizer support, ensure you have the required packages installed:

- `libasan` - AddressSanitizer runtime library
- `libasan-static` - Static version of AddressSanitizer runtime library

On Red Hat/Fedora systems:

```
sudo dnf install libasan libasan-static
```

Package names and versions may vary depending on your distribution and compiler version.

8.3 Debug Mode

When building in Debug mode, **pgmoneta** automatically enables various compiler flags to help with debugging, including AddressSanitizer (ASAN) and UndefinedBehaviorSanitizer (UBSAN) support when available.

The Debug mode can be enabled by adding `-DCMAKE_BUILD_TYPE=Debug` to your CMake command.

8.4 Sanitizer Flags

8.4.1 AddressSanitizer (ASAN)

Address Sanitizer is a memory error detector that helps find use-after-free, heap/stack/global buffer overflow, use-after-return, initialization order bugs, and memory leaks.

8.4.2 UndefinedBehaviorSanitizer (UBSAN)

UndefinedBehaviorSanitizer is a fast undefined behavior detector that can find various types of undefined behavior during program execution, such as integer overflow, null pointer dereference, and more.

8.4.3 Common Flags

- `-fno-omit-frame-pointer` - Provides better stack traces in error reports
- `-Wall -Wextra` - Enables additional compiler warnings

8.4.4 GCC Support

- `-fsanitize=address` - Enables the Address Sanitizer (GCC 4.8+)
- `-fsanitize=undefined` - Enables the Undefined Behavior Sanitizer (GCC 4.9+)
- `-fno-sanitize=alignment` - Disables alignment checking (GCC 5.1+)
- `-fno-sanitize-recover=all` - Makes all sanitizers halt on error (GCC 5.1+)
- `-fsanitize=float-divide-by-zero` - Detects floating-point division by zero (GCC 5.1+)
- `-fsanitize=float-cast-overflow` - Detects floating-point cast overflows (GCC 5.1+)
- `-fsanitize-recover=address` - Allows the program to continue execution after detecting an error (GCC 6.0+)
- `-fsanitize-address-use-after-scope` - Detects use-after-scope bugs (GCC 7.0+)

8.4.5 Clang Support

- `-fsanitize=address` - Enables the Address Sanitizer (Clang 3.2+)
- `-fno-sanitize=null` - Disables null pointer dereference checking (Clang 3.2+)
- `-fno-sanitize=alignment` - Disables alignment checking (Clang 3.2+)
- `-fsanitize=undefined` - Enables the Undefined Behavior Sanitizer (Clang 3.3+)
- `-fsanitize=float-divide-by-zero` - Detects floating-point division by zero (Clang 3.3+)
- `-fsanitize=float-cast-overflow` - Detects floating-point cast overflows (Clang 3.3+)
- `-fno-sanitize-recover=all` - Makes all sanitizers halt on error (Clang 3.6+)
- `-fsanitize-recover=address` - Allows the program to continue execution after detecting an error (Clang 3.8+)
- `-fsanitize-address-use-after-scope` - Detects use-after-scope bugs (Clang 3.9+)

8.5 Additional Sanitizer Options

Developers can add additional sanitizer flags via environment variables. Some useful options include:

8.5.1 ASAN Options

- `ASAN_OPTIONS=detect_leaks=1` - Enables memory leak detection
- `ASAN_OPTIONS=halt_on_error=0` - Continues execution after errors
- `ASAN_OPTIONS=detect_stack_use_after_return=1` - Enables stack use-after-return detection
- `ASAN_OPTIONS=check_initialization_order=1` - Detects initialization order problems
- `ASAN_OPTIONS=strict_string_checks=1` - Enables strict string function checking
- `ASAN_OPTIONS=detect_invalid_pointer_pairs=2` - Enhanced pointer pair validation
- `ASAN_OPTIONS=print_stats=1` - Prints statistics about allocated memory
- `ASAN_OPTIONS=verbosity=1` - Increases logging verbosity

8.5.2 UBSAN Options

- `UBSAN_OPTIONS=print_stacktrace=1` - Prints stack traces for errors
- `UBSAN_OPTIONS=halt_on_error=1` - Stops execution on the first error
- `UBSAN_OPTIONS=silence_unsigned_overflow=1` - Silences unsigned integer overflow reports

8.6 Building with Sanitizers

To build **pgmoneta** with sanitizer support:

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Debug ..
make
```

The compiler can also be specified

```
cmake -DCMAKE_C_COMPILER=gcc -DCMAKE_BUILD_TYPE=Debug ..
# or
cmake -DCMAKE_C_COMPILER=clang -DCMAKE_BUILD_TYPE=Debug .
```

The build system will automatically detect the compiler version and enable the appropriate sanitizer flags based on support.

8.7 Running with Sanitizers

When running **pgmoneta** built with sanitizers, any errors will be reported to stderr.

To get more detailed reports, you can set additional environment variables:

```
ASAN_OPTIONS=detect_leaks=1:halt_on_error=0:detect_stack_use_after_return
    =1 ./pgmoneta
```

You can combine ASAN and UBSAN options:

```
ASAN_OPTIONS=detect_leaks=1 UBSAN_OPTIONS=print_stacktrace=1 ./pgmoneta
```

8.8 Advanced Sanitizer Options Not Included by Default

Developers may want to experiment with additional sanitizer flags not enabled by default:

- `-fsanitize=memory` - Enables MemorySanitizer (MSan) for detecting uninitialized reads (Note this can't be used with ASan)
- `-fsanitize=integer` - Only check integer operations (subset of UBSan)
- `-fsanitize=bounds` - Array bounds checking (subset of UBSan)
- `-fsanitize-memory-track-origins` - Tracks origins of uninitialized values (with MSan)
- `-fsanitize-memory-use-after-dtor` - Detects use-after-destroy bugs (with MSan)
- `-fno-common` - Prevents variables from being merged into common blocks, helping identify variable access issues

Note that some sanitizers are incompatible with each other. For example, you cannot use ASan and MSan together.

9 Test

9.1 Local Environment

To ensure the test suite works well, please make sure you have installed PostgreSQL 17.x version installed

For RPM based distributions such as Fedora and RHEL you can add the PostgreSQL YUM repository and do the install via

```
dnf -qy module disable postgresql
dnf install -y postgresql17 postgresql17-server pgmoneta
```

also make sure that the `initdb`, `pg_ctl` and `psql` are in PATH variable.

9.1.1 Add Path variable

Add the `initdb`, `pg_ctl` and `psql` binaries into the environment path.

```
export PATH=$PATH:$(dirname $(which initdb))
export PATH=$PATH:$(dirname $(which psql))
```

Note: `initdb` and `pg_ctl` belongs to same binary directory

9.1.2 Install check library

Before you test, you need to install the `check` library. If there is no package for `check`, the `CMakeLists.txt` will not compile the test suite. Only after you have installed `check` will it compile the test suite.

```
dnf install -y check check-devel check-static
```

9.1.3 Build the project

Make sure to execute the test script inside the project build. Run the following commands if project is not already built.

```
git clone https://github.com/pgmoneta/pgmoneta.git
cd pgmoneta
mkdir build
cd build
cmake -DCMAKE_C_COMPILER=clang -DCMAKE_BUILD_TYPE=Debug ..
make
```

You can do

```
cmake -DCMAKE_C_COMPILER=clang -DCMAKE_BUILD_TYPE=Debug -DCMAKE_C_FLAGS="-
    DCORE_DEBUG" ..
```

in order to get information from the core libraries too.

9.1.4 Run test suites

To run the testsuites get inside your build and just execute -

```
./testsuite.sh
```

The script creates the PostgreSQL and pgmoneta environment inside the build itself for example - - the PostgreSQL related files like the data directory and PostgreSQL configuration will be stored in `pgmoneta-postgres` - the pgmoneta related files like pgmoneta configuration and users file will be stored in `pgmoneta-testsiute`

It will be the responsibility of the script to clean up the setup environment.

Note: You can however view the PostgreSQL and pgmoneta server logs in a separate `log` directory inside build.

In case you see those setup directories like `pgmoneta-postgres` and `pgmoneta-testsiute` in build after successfully executing the script, you should probably run

```
./testsuite clean
```

before running the script again to avoid any inconsistency or errors. The clean subcommand will however clean the logs as well.

9.1.5 Add testcases

To add an additional testcase go to testcases directory inside the `pgmoneta` project.

Create a `.c` file that contains the test suite and its corresponding `.h` file (see pgmoneta_test_1.c or pgmoneta_test_2.c for reference). Add the above created suite to the test runner in runner.c

Also remember to link the new test suite in CMakeLists file inside test directory

```
set(SOURCES
  testcases/common.c
  testcases/pgmoneta_test_1.c
  testcases/pgmoneta_test_2.c
  testcases/runner.c
)
```

9.1.6 Running Containerized Tests

The test suite supports containerized testing environments. When you run the C tests in the `build` directory, a Docker (or Podman) container is automatically started, and the test script is executed inside it. This ensures a consistent and isolated environment for testing.

> **Note:** The containerized test option (`ctest`) is only available if Docker or Podman is installed on your system. The CMake configuration will detect this and enable the container test target accordingly.

You have two main options to run the tests:

9.1.6.1 1. Using CTest (with Docker/Podman)

From the `build` directory, simply run:

```
ctest -V
```

This will: - Spin up the Docker/Podman container - Execute all test scripts inside the container - Collect logs, coverage, and test outputs

9.1.6.2 2. Using coverage.sh

Alternatively, you can run the coverage script directly:

```
./coverage.sh
```

This will: - Run all tests in the container - Generate code coverage reports

9.1.6.3 Artifacts

After running the tests, you will find: - **Test logs:** `build/log/` - **Coverage reports:** `build/coverage/` - **CTest logs:** `build/testing/`

10 Code Coverage

10.1 Automatic Code Coverage Detection

If both `gcov` and `gcovr` are installed on your system **and the compiler is set to GCC** (regardless of the build type), code coverage will be automatically enabled during the build process. The build scripts will detect these tools and set the appropriate flags. If either tool is missing, or if the compiler is not GCC, code coverage will be skipped and a message will indicate that coverage tools were not found or the compiler is not supported.

10.2 Generating Coverage Reports

After building the project with coverage enabled and running your testsuite, coverage reports will be generated automatically in the `build/coverage` directory if `gcov` and `gcovr` are available.

The following commands are used to generate the reports (executed automatically by the test scripts):

```
# Make sure the coverage directory exists
mkdir -p ./coverage

gcovr -r ../src --object-directory . --html --html-details -o ./coverage/
    index.html
gcovr -r ../src --object-directory . > ./coverage/summary.txt
```

Important:
These commands must be run from inside the `build` directory.

- The HTML report will be available at `build/coverage/index.html`
- A summary text report will be available at `build/coverage/summary.txt`

If you want to generate these reports manually after running your own tests, simply run the above commands from your `build` directory.

> **Note:** If the `coverage` directory does not exist, create it first using `mkdir -p ./coverage` before running the coverage commands.
>
> **Important:** `gcovr` only works with GCC builds. If you build the project with Clang, coverage reports will not be generated with `gcovr`.

10.3 Notes

- Make sure you have both `gcov` and `gcovr` installed before building the project to enable coverage.
- If coverage tools are not found, or the compiler is not GCC, coverage generation will be skipped automatically and a message will be shown.
- You can always re-run the coverage commands manually if needed.

11 WAL Reader

11.1 Overview

This document provides an overview of the wal_reader tool, with a focus on the parse_wal_file function, which serves as the main entry point for parsing Write-Ahead Log (WAL) files. Currently, the function only parses the given WAL file and prints the description of each record. In the future, it will be integrated with other parts of the code.

11.2 pgmoneta-walinfo

pgmoneta-walinfo is a command line utility designed to read and display information about PostgreSQL Write-Ahead Log (WAL) files. The tool provides output in either raw or JSON format, making it easy to analyze WAL files for debugging, auditing, or general information purposes.

In addition to standard WAL files, pgmoneta-walinfo also supports encrypted (**aes**) and compressed WAL files in the following formats: **zstd**, **gz**, **lz4**, and **bz2**.

11.2.0.1 Usage

```
pgmoneta-walinfo 0.18.0
  Command line utility to read and display Write-Ahead Log (WAL) files

Usage:
  pgmoneta-walinfo <file>

Options:
  -c,  --config      Set the path to the pgmoneta_walinfo.conf file
  -u,  --users       Set the path to the pgmoneta_users.conf file
  -RT, --tablespaces Filter on tablspaces
  -RD, --databases   Filter on databases
  -RT, --relations   Filter on relations
  -R,  --filter      Combination of -RT, -RD, -RR
  -o,  --output      Output file
  -F,  --format      Output format (raw, json)
  -L,  --logfile     Set the log file
  -q,  --quiet       No output only result
       --color       Use colors (on, off)
  -r,  --rmgr        Filter on a resource manager
  -s,  --start       Filter on a start LSN
  -e,  --end         Filter on an end LSN
  -x,  --xid         Filter on an XID
  -l,  --limit       Limit number of outputs
  -v,  --verbose     Output result
  -S,  --summary     Show a summary of WAL record counts grouped by
      resource manager
  -V,  --version     Display version information
  -m,  --mapping     Provide mappings file for OID translation
  -t,  --translate   Translate OIDs to object names in XLOG records
  -?,  --help        Display help
```

11.2.0.2 Raw Output Format

In `raw` format, the default, the output is structured as follows:

```
Resource Manager | Start LSN | End LSN | rec len | tot len | xid |
    description (data and backup)
```

- **Resource Manager**: The name of the resource manager handling the log record.
- **Start LSN**: The start Log Sequence Number (LSN).
- **End LSN**: The end Log Sequence Number (LSN).
- **rec len**: The length of the WAL record.
- **tot len**: The total length of the WAL record, including the header.
- **xid**: The transaction ID associated with the record.
- **description (data and backup)**: A detailed description of the operation, along with any related backup block information.

Each part of the output is color-coded:

- **Red**: Header information (resource manager, record length, transaction ID, etc.).
- **Green**: Description of the WAL record.
- **Blue**: Backup block references or additional data.

This format makes it easy to visually distinguish different parts of the WAL file for quick analysis.

11.2.0.3 Example

1. To view WAL file details in JSON format:

```
pgmoneta-walinfo -F json /path/to/walfile
```

2. To view WAL file details with OIDs in the records translated to object names:

Currently, `pgmoneta-walinfo` supports translating OIDs in two ways, 1. If the user provided `pgmoneta_user.conf` file, the tool will use it to get the needed credentials to connect to the database cluster and fetch the object names. directly from it.

```
pgmoneta-walinfo -c pgmoneta_walinfo.conf -t -u /path/to/pgmoneta_user.
    conf /path/to/walfile
```

2. If the user provided a mapping file that contains the OIDs and the corresponding object names, the tool will use it to translate the OIDs to the object names. This option helps if the user doesn't have the `pgmoneta_user.conf` file or doesn't want to use it.

```
pgmoneta-walinfo -c pgmoneta_walinfo.conf -t -m /path/to/mapping.json /
    path/to/walfile
```

User can get the needed info to create the file using these queries:

```
SELECT spcname, oid FROM pg_tablespace
SELECT datname, oid FROM pg_database
SELECT nspname || '.' || relname, c.oid FROM pg_class c JOIN pg_namespace
    n ON c.relnamespace = n.oid
```

In either ways, the user should use the `-t` flag to enable the translation. If user provided `pgmoneta_user.conf` file or the mapping file, the tool will do nothing if the `-t` flag is not provided.

User can create the `pgmoneta_user.conf` file by following the instructions in the `DEVELOPER.md`

After using this translation feature, the output will change XLOG records from something like this `Heap2 | 1/D8FFD1C0 | 1/D8FFEB50 | 59 | 59 | 958 | cutoff xid 0 flags 0x03 blkref #0: rel 1663/16399/16733 forknum 2 blk 0 blkref #1: rel 1663/16399/16733 forknum 0 blk 27597`

to this `Heap2 | 1/D8FFD1C0 | 1/D8FFEB50 | 59 | 59 | 958 | cutoff xid 0 flags 0x03 blkref #0: rel pg_default/mydb/test_tbl forknum 2 blk 0 blkref #1: rel pg_default/mydb/16733 forknum 0 blk 27597`

Example of `mappings.json` file:

```
{
    "tablespaces": [
        {"name1": "oid1"},
        {"name2": "oid2"}
    ],
    "databases": [
        {"name1": "oid1"},
        {"name2": "oid2"}
    ],
    "relations": [
        {"name1": "oid1"},
        {"name2": "oid2"}
    ]
}
```

which is basically three sections, each section contains array key value pairs. The key is the object name and the value is the oid.

Note 1: If both files (`pgmoneta_users.conf` & `mappings.json`) are provided, the tool will use the mapping file. Note 2: If there is an OID that wasn't in the server/mapping (whichever the user choose at that time), the oid will be written as it is.

e.g. `rel pg_default/mydb/16733` will be written as `rel pg_default/mydb/16733` if the OID `16733` wasn't in the server/mapping.

11.3 High-Level API Overview

The following section provides a high-level overview of how users can interact with the functions and structures defined in the `walfile.h` file. These APIs allow you to read, write, and manage Write-Ahead Log (WAL) files.

11.3.1 Struct `walfile`

The `walfile` struct represents the core structure used for interacting with WAL files in PostgreSQL. A WAL file stores a log of changes to the database and is used for crash recovery, replication, and other purposes. Each WAL file consists of pages (each 8192 bytes by default), containing records that capture database changes.

11.3.1.1 Fields:

- **magic_number**: Identifies the PostgreSQL version that created the WAL file. You can find more info on supported magic numbers here.
- **long_phd**: A pointer to the extended header (long header) found on the first page of the WAL file. This header contains additional metadata.
- **page_headers**: A deque of headers representing each page in the WAL file, excluding the first page.
- **records**: A deque of decoded WAL records. Each record represents a change made to the database and contains both metadata and the actual data to be applied during recovery or replication.

11.3.2 Function Overview

The `walfile.h` file provides three key functions for interacting with WAL files: `pgmoneta_read_walfile`, `pgmoneta_write_walfile`, and `pgmoneta_destroy_walfile`. These functions allow users to read from, write to, and destroy WAL file objects, respectively.

11.3.2.1 pgmoneta_read_walfile

```
int pgmoneta_read_walfile(int server, char* path, struct walfile** wf);
```

11.3.2.1.1 Description:

This function reads a WAL file from a specified path and populates a walfile structure with its contents, including the file's headers and records.

11.3.2.1.2 Parameters:

- **server**: The index of the Postgres server in Pgmoneta configuration.
- **path**: The file path to the WAL file that needs to be read.
- **wf**: A pointer to a pointer to a walfile structure that will be populated with the WAL file data.

11.3.2.1.3 Return:

- Returns 0 on success or 1 on failure.

11.3.2.1.4 Usage Example:

```
struct walfile* wf = NULL;
int result = pgmoneta_read_walfile(0, "/path/to/walfile", &wf);
if (result == 0) {
    // Successfully read WAL file
}
```

11.3.2.2 pgmoneta_write_walfile

```
int pgmoneta_write_walfile(struct walfile* wf, int server, char* path);
```

11.3.2.2.1 Description:

This function writes the contents of a walfile structure back to disk, saving it as a WAL file at the specified path.

11.3.2.2.2 Parameters:

- **wf**: The walfile structure containing the WAL data to be written.
- **server**: The index or ID of the server where the WAL file should be saved.
- **path**: The file path where the WAL file should be written.

11.3.2.2.3 Return:

- Returns 0 on success or 1 on failure.

11.3.2.2.4 Usage Example:

```
int result = pgmoneta_write_walfile(wf, 0, "/path/to/output_walfile");
if (result == 0) {
    // Successfully wrote WAL file
}
```

11.3.2.3 pgmoneta_destroy_walfile

```
void pgmoneta_destroy_walfile(struct walfile* wf);
```

11.3.2.3.1 Description: This function frees the memory allocated for a `walfile` structure, including its headers and records.

11.3.2.3.2 Parameters:

- **wf**: The `walfile` structure to be destroyed.

11.3.2.3.3 Usage Example:

```
struct walfile* wf = NULL;
int result = pgmoneta_read_walfile(0, "/path/to/walfile", &wf);
if (result == 0) {
    // Successfully read WAL file
}
pgmoneta_destroy_walfile(wf);
```

11.3.2.4 pgmoneta_describe_walfile

```
int pgmoneta_describe_walfile(char* path, enum value_type type, FILE*
    output, bool quiet, bool color,
                                struct deque* rms, uint64_t start_lsn,
                                    uint64_t end_lsn, struct deque* xids,
                                uint32_t limit, bool summary, char**
                                    included_objects);
```

11.3.2.4.1 Description: This function reads a single WAL file at the specified `path`, filters its records based on provided parameters, and writes the formatted output to `output`.

11.3.2.4.2 Parameters:

- **path**: Path to the WAL file to be described
- **type**: Output format type (raw or JSON)
- **output**: File stream for output; if NULL, prints to stdout
- **quiet**: If true, suppresses detailed output

- **color**: If true, enables colored output for better readability
- **rms**: Deque of resource managers to filter on
- **start_lsn**: Starting LSN to filter records (0 for no filter)
- **end_lsn**: Ending LSN to filter records (0 for no filter)
- **xids**: Deque of transaction IDs to filter on
- **limit**: Maximum number of records to output (0 for no limit)
- **summary**: Show a summary of WAL record counts grouped by resource manager
- **included_objects**: Array of object names to filter on (NULL for all objects)

11.3.2.4.3 Return:

- Returns `0` on success or `1` on failure.

```
#### ‘pgmoneta_describe_walfiles_in_directory‘

‘‘‘c
int pgmoneta_describe_walfiles_in_directory(char* dir_path, enum
    value_type type, FILE* output,
                                             bool quiet, bool color, struct
                                                  deque* rms,
                                             uint64_t start_lsn, uint64_t
                                                 end_lsn, struct deque* xids
                                                 ,
                                             uint32_t limit, bool summary,
                                                 char** included_objects);
```

11.3.2.4.4 Description:

This function processes all WAL files in the directory specified by `dir_path` , applies the same filtering logic as `pgmoneta_describe_walfile`, and writes aggregated results to `output`.

11.3.2.4.5 Parameters:

- **dir_path**: Path to the directory containing WAL files
- **type**: Output format type (raw or JSON)
- **output**: File stream for output; if NULL, prints to stdout
- **quiet**: If true, suppresses detailed output
- **color**: If true, enables colored output for better readability
- **rms**: Deque of resource managers to filter on
- **start_lsn**: Starting LSN to filter records (0 for no filter)
- **end_lsn**: Ending LSN to filter records (0 for no filter)

- **xids**: Deque of transaction IDs to filter on
- **limit**: Maximum number of records to output (0 for no limit)
- **summary**: Show a summary of WAL record counts grouped by resource manager
- **included_objects**: Array of object names to filter on (NULL for all objects)

11.3.2.4.6 Return:

- Returns `0` on success or `1` on failure

11.4 Internal API Overview

11.4.1 struct partial_xlog_record

The partial_xlog_record struct represents an incomplete WAL XLOG record encountered during parsing. It is used to manage records that span multiple WAL files.

```
struct partial_xlog_record
{
   char* data_buffer;                   /**< Data portion of the record. */
   char* xlog_record;                   /**< Pointer to the xlog record. */
   uint32_t data_buffer_bytes_read;     /**< Length of the total data read
      in data_buffer. */
   uint32_t xlog_record_bytes_read;     /**< Length of the total data read
      in xlog_record buffer. */
};
```

11.4.1.0.1 Fields:

- **data_buffer**: Contains the data portion of the partially read WAL record.
- **xlog_record**: Points to the header structure containing metadata about the WAL record.
- **data_buffer_bytes_read**: Length of the total data read in data_buffer.
- **xlog_record_bytes_read**: Length of the total data read in xlog_record buffer.

11.4.2 parse_wal_file

This function is responsible for reading and parsing a PostgreSQL Write-Ahead Log (WAL) file.

11.4.2.1 Parameters

- **path**: The file path to the WAL file that needs to be parsed.
- **server_info**: A pointer to a `server` structure containing information about the server.

11.4.2.2 Description

The `parse_wal_file` function opens the WAL file specified by the `path` parameter in binary mode and reads the WAL records. It processes these records, handling various cases such as records that cross page boundaries, while ensuring correct memory management throughout the process.

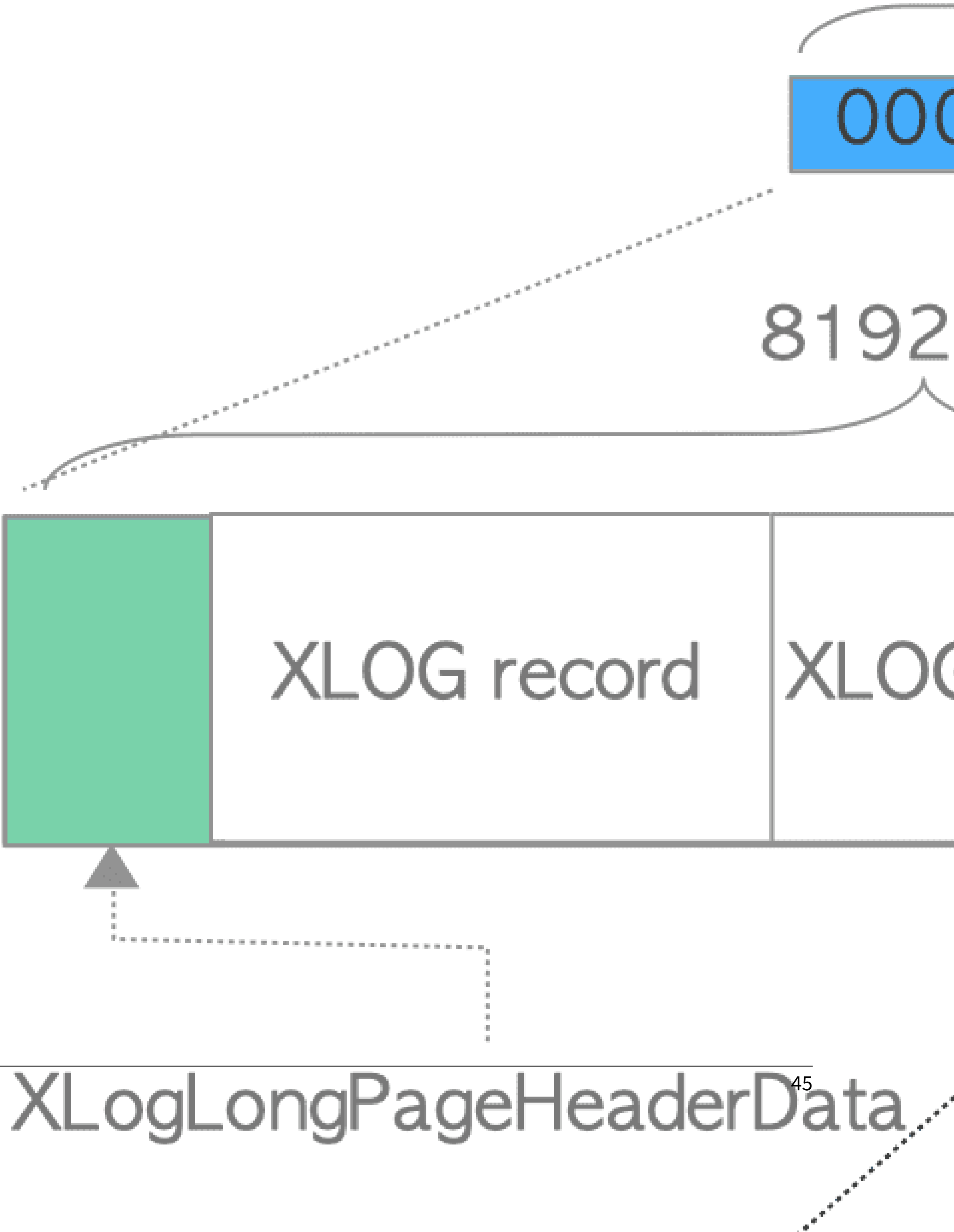
11.4.3 Usage Example

```
parse_wal_file("/path/to/wal/file", &my_server);
```

11.4.4 WAL File Structure

The image illustrates the structure of a WAL (Write-Ahead Logging) file in PostgreSQL, focusing on how XLOG records are organized within WAL segments.

Source: https://www.interdb.jp/pg/pgsql09/03.html


000
8192
XLOG record
XLOG
XLogLongPageHeaderData

A WAL segment, by default, is a 16 MB file, divided into pages of 8192 bytes (8 KB) each. The first page contains a header defined by the XLogLongPageHeaderData structure, while all subsequent pages have headers described by the XLogPageHeaderData structure. XLOG records are written sequentially in each page, starting at the beginning and moving downward.

The figure highlights how the WAL ensures data consistency by sequentially writing XLOG records in pages, structured within larger 16 MB WAL segments.

11.5 Resource Managers

In the context of the WAL reader, resource managers (rm) are responsible for handling different types of records found within a WAL file. Each record in the WAL file is associated with a specific resource manager, which determines how that record is processed.

11.5.1 Resource Manager Definitions

Each resource manager is defined in the `rm_[name].h` header file and implemented in the corresponding `rm_[name].c` source file.

In the `rmgr.h` header file, the resource managers are declared as an enum, with each resource manager having a unique identifier.

11.5.2 Resource Manager Functions

Each resource manager implements the `rm_desc` function, which provides a description of the record type associated with that resource manager. In the future, they will be extended to implement the `rm_redo` function to apply the changes to another server.

11.5.3 Supporting Various WAL Structures in PostgreSQL Versions 13 to 17

The WAL structure has evolved across PostgreSQL versions 13 to 17, requiring different handling for each version. To accommodate these differences, we have implemented a wrapper-based approach, such as the factory pattern, to handle varying WAL structures.

Below are the commit hashes for the officially supported magic values in each PostgreSQL version:

1. PostgreSQL 13 - 0xD106: https://github.com/postgres/postgres/commit/c6b92041d38512a4176ed76ad06f713d2e
2. PostgreSQL 14 - 0xD10D: https://github.com/postgres/postgres/commit/08aa89b326261b669648df97d4f2a6edba
3. PostgreSQL 15 - 0xD110: https://github.com/postgres/postgres/commit/8b1dccd37c71ed2ff016294d8f9053a32b0

4. PostgreSQL 16 - 0xD113: https://github.com/postgres/postgres/commit/6af1793954e8c5e753af83c3edb37ed3267
5. PostgreSQL 17 - 0xD116: https://github.com/postgres/postgres/commit/402b586d0a9caae9412d25fcf1b91dae45
6. PostgreSQL 18 - 0xD118: https://github.com/postgres/postgres/commit/243e9b40f1b2dd09d6e5bf91ebf6e822a2

`xl_end_of_recovery` is an example of how we handle different versions of structures with a wrapper struct and a factory pattern.

```c
struct xl_end_of_recovery_v16 {
    timestamp_tz end_time;
    timeline_id this_timeline_id;
    timeline_id prev_timeline_id;
};

struct xl_end_of_recovery_v17 {
    timestamp_tz end_time;
    timeline_id this_timeline_id;
    timeline_id prev_timeline_id;
    int wal_level;
};

struct xl_end_of_recovery {
    int pg_version;
    union {
        struct xl_end_of_recovery_v16 v16;
        struct xl_end_of_recovery_v17 v17;
    } data;
    void (*parse)(struct xl_end_of_recovery* wrapper, void* rec);
    char* (*format)(struct xl_end_of_recovery* wrapper, char* buf);
};

xl_end_of_recovery* create_xl_end_of_recovery(int pg_version) {
    xl_end_of_recovery* wrapper = malloc(sizeof(xl_end_of_recovery));
    wrapper->pg_version = pg_version;

    if (pg_version >= 17) {
        wrapper->parse = parse_v17;
        wrapper->format = format_v17;
    } else {
        wrapper->parse = parse_v16;
        wrapper->format = format_v16;
    }

    return wrapper;
}

void parse_v16(xl_end_of_recovery* wrapper, void* rec) {
    memcpy(&wrapper->data.v16, rec, sizeof(struct xl_end_of_recovery_v16))
       ;
}
```

```
void parse_v17(xl_end_of_recovery* wrapper, void* rec) {
    memcpy(&wrapper->data.v17, rec, sizeof(struct xl_end_of_recovery_v17))
        ;
}

char* format_v16(xl_end_of_recovery* wrapper, char* buf) {
    struct xl_end_of_recovery_v16* xlrec = &wrapper->data.v16;
    return pgmoneta_format_and_append(buf, "tli %u; prev tli %u; time %s",
                                      xlrec->this_timeline_id, xlrec->
                                         prev_timeline_id,
                                      pgmoneta_wal_timestamptz_to_str(
                                         xlrec->end_time));
}

char* format_v17(xl_end_of_recovery* wrapper, char* buf) {
    struct xl_end_of_recovery_v17* xlrec = &wrapper->data.v17;
    return pgmoneta_format_and_append(buf, "tli %u; prev tli %u; time %s;
       wal_level %d",
                                      xlrec->this_timeline_id, xlrec->
                                         prev_timeline_id,
                                      pgmoneta_wal_timestamptz_to_str(
                                         xlrec->end_time),
                                      xlrec->wal_level);
}
```

11.6 WAL Change List

This section lists the changes in the WAL format between different versions of PostgreSQL.

11.6.1 xl_clog_truncate

17

```
struct xl_clog_truncate
{
   int64 pageno;                /**< The page number of the clog to truncate
        */
   transaction_id oldestXact;   /**< The oldest transaction ID to retain */
   oid oldestXactDb;          /**< The database ID of the oldest transaction
        */
};
```

16

```
struct xl_clog_truncate
{
   int64 pageno;               /**< The page number of the clog to truncate
       */
   transaction_id oldestXact;  /**< The oldest transaction ID to retain */
   oid oldestXactDb;         /**< The database ID of the oldest transaction
       */
};
```

11.6.2 xl_commit_ts_truncate

17:

```
typedef struct xl_commit_ts_truncate
{
    int64       pageno;
    TransactionId oldestXid;
} xl_commit_ts_truncate;
```

16:

```
typedef struct xl_commit_ts_truncate
{
    int         pageno;
    TransactionId oldestXid;
} xl_commit_ts_truncate;
```

11.6.3 xl_heap_prune

17:

```
typedef struct xl_heap_prune
{
    uint8       reason;
    uint8       flags;

    /*
     * If XLHP_HAS_CONFLICT_HORIZON is set, the conflict horizon XID
        follows,
     * unaligned
     */
} xl_heap_prune;
#define SizeOfHeapPrune (offsetof(xl_heap_prune, flags) + sizeof(uint8))
```

16:

```
typedef struct xl_heap_prune
{
    TransactionId snapshotConflictHorizon;
    uint16      nredirected;
    uint16      ndead;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_prune;
#define SizeOfHeapPrune (offsetof(xl_heap_prune, isCatalogRel) + sizeof(
    bool))
```

11.6.4 xlhp_freeze_plan

Removed `xl_heap_freeze_page`

17:

```
typedef struct xlhp_freeze_plan
{
    TransactionId xmax;
    uint16      t_infomask2;
    uint16      t_infomask;
    uint8       frzflags;

    /* Length of individual page offset numbers array for this plan */
    uint16      ntuples;
} xlhp_freeze_plan;
```

11.6.5 spgxlogState

(Doesn't need to be changed)

17:

```
typedef struct spgxlogState
{
    TransactionId redirectXid;
    bool        isBuild;
} spgxlogState;
```

16:

```
typedef struct spgxlogState
{
    TransactionId myXid;
    bool        isBuild;
} spgxlogState;
```

11.6.6 xl_end_of_recovery

```
typedef struct xl_end_of_recovery
{
    TimestampTz end_time;
    TimeLineID  ThisTimeLineID; /* new TLI */
    TimeLineID  PrevTimeLineID; /* previous TLI we forked off from */
    int         wal_level;
} xl_end_of_recovery;
```

16:

```
typedef struct xl_end_of_recovery
{
    TimestampTz end_time;
    TimeLineID  ThisTimeLineID; /* new TLI */
    TimeLineID  PrevTimeLineID; /* previous TLI we forked off from */
} xl_end_of_recovery;
```

---

16 → 15

11.6.7 gingxlogSplit

16: same for `gin_xlog_update_meta`

```
typedef struct ginxlogSplit
{
    RelFileLocator locator;
    BlockNumber rrlink;         /* right link, or root's blocknumber if
       root
                                 * split */
    BlockNumber leftChildBlkno; /* valid on a non-leaf split */
    BlockNumber rightChildBlkno;
    uint16      flags;          /* see below */
} ginxlogSplit;
```

15:

```
typedef struct ginxlogSplit
{
    RelFileNode node;
    BlockNumber rrlink;         /* right link, or root's blocknumber if
       root
                                 * split */
    BlockNumber leftChildBlkno; /* valid on a non-leaf split */
    BlockNumber rightChildBlkno;
    uint16      flags;          /* see below */
} ginxlogSplit;
```

11.6.8 gistxlogDelete

16:

```
typedef struct gistxlogDelete
{
    TransactionId snapshotConflictHorizon;
    uint16      ntodelete;      /* number of deleted offsets */
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */

    /* TODELETE OFFSET NUMBERS */
    OffsetNumber offsets[FLEXIBLE_ARRAY_MEMBER];
} gistxlogDelete;
#define SizeOfGistxlogDelete    offsetof(gistxlogDelete, offsets)
```

15:

```
typedef struct gistxlogDelete
{
    TransactionId latestRemovedXid;
    uint16      ntodelete;      /* number of deleted offsets */

    /*
     * In payload of blk 0 : todelete OffsetNumbers
     */
} gistxlogDelete;
#define SizeOfGistxlogDelete    (offsetof(gistxlogDelete, ntodelete) +
   sizeof(uint16))
```

11.6.9 gistxlogPageReuse

16:

```
typedef struct gistxlogPageReuse
{
    RelFileLocator locator;
    BlockNumber block;
    FullTransactionId snapshotConflictHorizon;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */
} gistxlogPageReuse;
#define SizeOfGistxlogPageReuse (offsetof(gistxlogPageReuse, isCatalogRel)
     + sizeof(bool))
```

15:

```
typedef struct gistxlogPageReuse
{
    RelFileNode node;
    BlockNumber block;
    FullTransactionId latestRemovedFullXid;
} gistxlogPageReuse;

#define SizeOfGistxlogPageReuse (offsetof(gistxlogPageReuse,
   latestRemovedFullXid) + sizeof(FullTransactionId))
```

11.6.10 xl_hash_vacuum_one_page

16:

```
typedef struct xl_hash_vacuum_one_page
{
    TransactionId snapshotConflictHorizon;
    uint16      ntuples;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */

    /* TARGET OFFSET NUMBERS */
    OffsetNumber offsets[FLEXIBLE_ARRAY_MEMBER];
} xl_hash_vacuum_one_page;
#define SizeOfHashVacuumOnePage offsetof(xl_hash_vacuum_one_page, offsets)
```

15:

```
typedef struct xl_hash_vacuum_one_page
{
    TransactionId latestRemovedXid;
    int         ntuples;

    /* TARGET OFFSET NUMBERS FOLLOW AT THE END */
} xl_hash_vacuum_one_page;
#define SizeOfHashVacuumOnePage \
    (offsetof(xl_hash_vacuum_one_page, ntuples) + sizeof(int))
```

11.6.11 xl_heap_prune

16:

```
typedef struct xl_heap_prune
{
    TransactionId snapshotConflictHorizon;
    uint16      nredirected;
    uint16      ndead;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_prune;
#define SizeOfHeapPrune (offsetof(xl_heap_prune, isCatalogRel) + sizeof(
    bool))
```

15:

```
typedef struct xl_heap_prune
{
    TransactionId latestRemovedXid;
    uint16      nredirected;
    uint16      ndead;
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_prune;
#define SizeOfHeapPrune (offsetof(xl_heap_prune, ndead) + sizeof(uint16))
```

11.6.12 xl_heap_freeze_plan

16:

```
typedef struct xl_heap_freeze_plan
{
    TransactionId xmax;
    uint16      t_infomask2;
    uint16      t_infomask;
    uint8       frzflags;

    /* Length of individual page offset numbers array for this plan */
    uint16      ntuples;
} xl_heap_freeze_plan;
```

15:

```
typedef struct xl_heap_freeze_tuple
{
    TransactionId xmax;
    OffsetNumber offset;
    uint16      t_infomask2;
    uint16      t_infomask;
    uint8       frzflags;
} xl_heap_freeze_tuple;
```

11.6.13 xl_heap_freeze_page

16:

```
typedef struct xl_heap_freeze_page
{
    TransactionId snapshotConflictHorizon;
    uint16      nplans;
    bool        isCatalogRel;   /* to handle recovery conflict during
        logical
                                 * decoding on standby */

    /*
     * In payload of blk 0 : FREEZE PLANS and OFFSET NUMBER ARRAY
     */
} xl_heap_freeze_page;
```

15:

```
typedef struct xl_heap_freeze_page
{
    TransactionId cutoff_xid;
    uint16      ntuples;
} xl_heap_freeze_page;
```

11.6.14 xl_btree_reuse_page

16:

```
typedef struct xl_btree_reuse_page
{
    RelFileLocator locator;
    BlockNumber block;
    FullTransactionId snapshotConflictHorizon;
    bool        isCatalogRel;   /* to handle recovery conflict during
        logical
                                 * decoding on standby */
} xl_btree_reuse_page;
```

15:

```
typedef struct xl_btree_reuse_page
```

```
{
    RelFileNode node;
    BlockNumber block;
    FullTransactionId latestRemovedFullXid;
} xl_btree_reuse_page;
```

11.6.15 xl_btree_delete

16:

```
typedef struct xl_btree_delete
{
    TransactionId snapshotConflictHorizon;
    uint16      ndeleted;
    uint16      nupdated;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */

    /*----
     * In payload of blk 0 :
     * - DELETED TARGET OFFSET NUMBERS
     * - UPDATED TARGET OFFSET NUMBERS
     * - UPDATED TUPLES METADATA (xl_btree_update) ARRAY
     *----
     */
} xl_btree_delete;
```

15:

```
typedef struct xl_btree_delete
{
    TransactionId latestRemovedXid;
    uint16      ndeleted;
    uint16      nupdated;

    /* DELETED TARGET OFFSET NUMBERS FOLLOW */
    /* UPDATED TARGET OFFSET NUMBERS FOLLOW */
    /* UPDATED TUPLES METADATA (xl_btree_update) ARRAY FOLLOWS */
} xl_btree_delete;
```

11.6.16 spgxlogVacuumRedirect

16:

```
typedef struct spgxlogVacuumRedirect
{
    uint16      nToPlaceholder; /* number of redirects to make
       placeholders */
    OffsetNumber firstPlaceholder;  /* first placeholder tuple to remove
       */
    TransactionId snapshotConflictHorizon;  /* newest XID of removed
       redirects */
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */

    /* offsets of redirect tuples to make placeholders follow */
    OffsetNumber offsets[FLEXIBLE_ARRAY_MEMBER];
} spgxlogVacuumRedirect;
```

15:

```
typedef struct spgxlogVacuumRedirect
{
    uint16      nToPlaceholder; /* number of redirects to make
       placeholders */
    OffsetNumber firstPlaceholder;  /* first placeholder tuple to remove
       */
    TransactionId newestRedirectXid;    /* newest XID of removed redirects
        */

    /* offsets of redirect tuples to make placeholders follow */
    OffsetNumber offsets[FLEXIBLE_ARRAY_MEMBER];
} spgxlogVacuumRedirect;
```

---

15 → 14

11.6.17 xl_xact_prepare

15:

```
ctypedef struct xl_xact_prepare
{
    uint32      magic;          /* format identifier */
    uint32      total_len;      /* actual file length */
    TransactionId xid;          /* original transaction XID */
    Oid         database;       /* OID of database it was in */
    TimestampTz prepared_at;    /* time of preparation */
    Oid         owner;          /* user running the transaction */
    int32       nsubxacts;      /* number of following subxact XIDs */
    int32       ncommitrels;    /* number of delete-on-commit rels */
    int32       nabortrels;     /* number of delete-on-abort rels */
    int32       ncommitstats;   /* number of stats to drop on commit */
    int32       nabortstats;    /* number of stats to drop on abort */
    int32       ninvalmsgs;     /* number of cache invalidation messages
       */
    bool        initfileinval;  /* does relcache init file need
       invalidation? */
    uint16      gidlen;         /* length of the GID - GID follows the
       header */
    XLogRecPtr  origin_lsn;     /* lsn of this record at origin node */
    TimestampTz origin_timestamp;   /* time of prepare at origin node */
} xl_xact_prepare;
```

14:

```
typedef struct xl_xact_prepare
{
    uint32      magic;          /* format identifier */
    uint32      total_len;      /* actual file length */
    TransactionId xid;          /* original transaction XID */
    Oid         database;       /* OID of database it was in */
    TimestampTz prepared_at;    /* time of preparation */
    Oid         owner;          /* user running the transaction */
    int32       nsubxacts;      /* number of following subxact XIDs */
    int32       ncommitrels;    /* number of delete-on-commit rels */
    int32       nabortrels;     /* number of delete-on-abort rels */
    int32       ninvalmsgs;     /* number of cache invalidation messages
       */
    bool        initfileinval;  /* does relcache init file need
       invalidation? */
    uint16      gidlen;         /* length of the GID - GID follows the
       header */
    XLogRecPtr  origin_lsn;     /* lsn of this record at origin node */
    TimestampTz origin_timestamp;   /* time of prepare at origin node */
} xl_xact_prepare;
```

11.6.18 xl_xact_parsed_commit

15:

```
typedef struct xl_xact_parsed_commit
{
    TimestampTz xact_time;
    uint32      xinfo;

    Oid         dbId;           /* MyDatabaseId */
    Oid         tsId;           /* MyDatabaseTableSpace */

    int         nsubxacts;
    TransactionId *subxacts;
    int         nrels;
    RelFileNode *xnodes;

    int         nstats;
    xl_xact_stats_item *stats;

    int         nmsgs;
    SharedInvalidationMessage *msgs;

    TransactionId twophase_xid; /* only for 2PC */
    char        twophase_gid[GIDSIZE];  /* only for 2PC */
    int         nabortrels;     /* only for 2PC */
    RelFileNode *abortnodes;    /* only for 2PC */
    int         nabortstats;        /* only for 2PC */
    xl_xact_stats_item *abortstats; /* only for 2PC */

    XLogRecPtr  origin_lsn;
    TimestampTz origin_timestamp;
} xl_xact_parsed_commit;
```

14:

```
typedef struct xl_xact_parsed_commit
{
    TimestampTz xact_time;
    uint32      xinfo;

    Oid         dbId;           /* MyDatabaseId */
    Oid         tsId;           /* MyDatabaseTableSpace */

    int         nsubxacts;
    TransactionId *subxacts;

    int         nrels;
    RelFileNode *xnodes;

    int         nmsgs;
    SharedInvalidationMessage *msgs;

    TransactionId twophase_xid; /* only for 2PC */
    char        twophase_gid[GIDSIZE];  /* only for 2PC */
    int         nabortrels;     /* only for 2PC */
    RelFileNode *abortnodes;    /* only for 2PC */

    XLogRecPtr  origin_lsn;
    TimestampTz origin_timestamp;
} xl_xact_parsed_commit;
```

11.6.19 xl_xact_parsed_abort

15:

```
typedef struct xl_xact_parsed_abort
{
    TimestampTz xact_time;
    uint32      xinfo;

    Oid         dbId;           /* MyDatabaseId */
    Oid         tsId;           /* MyDatabaseTableSpace */

    int         nsubxacts;
    TransactionId *subxacts;

    int         nrels;
    RelFileNode *xnodes;

    int         nstats;
    xl_xact_stats_item *stats;

    TransactionId twophase_xid; /* only for 2PC */
    char        twophase_gid[GIDSIZE];  /* only for 2PC */

    XLogRecPtr  origin_lsn;
    TimestampTz origin_timestamp;
} xl_xact_parsed_abort;
```

14:

```
typedef struct xl_xact_parsed_abort
{
    TimestampTz xact_time;
    uint32      xinfo;

    Oid         dbId;           /* MyDatabaseId */
    Oid         tsId;           /* MyDatabaseTableSpace */

    int         nsubxacts;
    TransactionId *subxacts;

    int         nrels;
    RelFileNode *xnodes;

    TransactionId twophase_xid; /* only for 2PC */
    char        twophase_gid[GIDSIZE];  /* only for 2PC */

    XLogRecPtr  origin_lsn;
    TimestampTz origin_timestamp;
} xl_xact_parsed_abort;
```

11.6.20 xlogrecord.h flags

15:

```
#define BKPIMAGE_APPLY          0x02    /* page image should be restored
                                         * during replay */
/* compression methods supported */
#define BKPIMAGE_COMPRESS_PGLZ  0x04
#define BKPIMAGE_COMPRESS_LZ4   0x08
#define BKPIMAGE_COMPRESS_ZSTD  0x10

#define BKPIMAGE_COMPRESSED(info) \
    ((info & (BKPIMAGE_COMPRESS_PGLZ | BKPIMAGE_COMPRESS_LZ4 | \
              BKPIMAGE_COMPRESS_ZSTD)) != 0)
```

14:

```
#define BKPIMAGE_IS_COMPRESSED      0x02    /* page image is compressed */
#define BKPIMAGE_APPLY      0x04    /* page image should be restored
   during
                                     * replay */
```

---

14 → 13

11.6.21 xl_heap_prune

14:

```
typedef struct xl_heap_prune
{
    TransactionId latestRemovedXid;
    uint16      nredirected;
    uint16      ndead;
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_prune;
```

13:

```
typedef struct xl_heap_clean
{
    TransactionId latestRemovedXid;
    uint16      nredirected;
    uint16      ndead;
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_clean;
```

11.6.22 xl_heap_vacuum

14:

```
typedef struct xl_heap_vacuum
{
    uint16      nunused;
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_vacuum;
```

13:

```
typedef struct xl_heap_cleanup_info
{
    RelFileNode node;
    TransactionId latestRemovedXid;
} xl_heap_cleanup_info;
```

11.6.23 xl_btree_metadata

14:

```
typedef struct xl_btree_metadata
{
    uint32      version;
    BlockNumber root;
    uint32      level;
    BlockNumber fastroot;
    uint32      fastlevel;
    uint32      last_cleanup_num_delpages;
    bool        allequalimage;
} xl_btree_metadata;
```

13:

```
typedef struct xl_btree_metadata
{
    uint32      version;
    BlockNumber root;
    uint32      level;
    BlockNumber fastroot;
    uint32      fastlevel;
    TransactionId oldest_btpo_xact;
    float8      last_cleanup_num_heap_tuples;
    bool        allequalimage;
} xl_btree_metadata;
```

11.6.24 xl_btree_reuse_page

14:

```c
typedef struct xl_btree_reuse_page
{
    RelFileNode node;
    BlockNumber block;
    FullTransactionId latestRemovedFullXid;
} xl_btree_reuse_page;
```

13:

```c
typedef struct xl_btree_reuse_page
{
    RelFileNode node;
    BlockNumber block;
    TransactionId latestRemovedXid;
} xl_btree_reuse_page;
```

11.6.25 xl_btree_delete

14:

```c
typedef struct xl_btree_delete
{
    TransactionId latestRemovedXid;
    uint16      ndeleted;
    uint16      nupdated;

    /* DELETED TARGET OFFSET NUMBERS FOLLOW */
    /* UPDATED TARGET OFFSET NUMBERS FOLLOW */
    /* UPDATED TUPLES METADATA (xl_btree_update) ARRAY FOLLOWS */
} xl_btree_delete;
```

13:

```c
typedef struct xl_btree_delete
{
    TransactionId latestRemovedXid;
    uint32      ndeleted;

    /* DELETED TARGET OFFSET NUMBERS FOLLOW */
} xl_btree_delete;
```

11.6.26 xl_btree_unlink_page

14:

```
typedef struct xl_btree_unlink_page
{
    BlockNumber leftsib;        /* target block's left sibling, if any */
    BlockNumber rightsib;       /* target block's right sibling */
    uint32      level;          /* target block's level */
    FullTransactionId safexid;  /* target block's BTPageSetDeleted() XID
       */

    /*
     * Information needed to recreate a half-dead leaf page with correct
     * topparent link.  The fields are only used when deletion operation's
     * target page is an internal page.  REDO routine creates half-dead
        page
     * from scratch to keep things simple (this is the same convenient
     * approach used for the target page itself).
     */
    BlockNumber leafleftsib;
    BlockNumber leafrightsib;
    BlockNumber leaftopparent;  /* next child down in the subtree */

    /* xl_btree_metadata FOLLOWS IF XLOG_BTREE_UNLINK_PAGE_META */
} xl_btree_unlink_page;
```

13:

```
typedef struct xl_btree_unlink_page
{
    BlockNumber leftsib;        /* target block's left sibling, if any */
    BlockNumber rightsib;       /* target block's right sibling */

    /*
     * Information needed to recreate the leaf page, when target is an
     * internal page.
     */
    BlockNumber leafleftsib;
    BlockNumber leafrightsib;
    BlockNumber topparent;      /* next child down in the branch */

    TransactionId btpo_xact;    /* value of btpo.xact for use in recovery
       */
    /* xl_btree_metadata FOLLOWS IF XLOG_BTREE_UNLINK_PAGE_META */
} xl_btree_unlink_page;
```

11.7 Additional Information

For more details on the internal workings and additional helper functions used in `parse_wal_file`, refer to the source code in `wal_reader.c`.

12 Core APIs

pgmoneta offers data structures and APIs to help you write safer code and enable you to develop more advanced functionalities. Currently, we offer adaptive radix tree (ART), deque and JSON, which are all based on a universal value type system that help you manage the memory easily.

The document will mostly focus on design decisions, functionalities and things to be cautious about. It may offer some examples as to how to use the APIs.

12.1 Value

The `value` struct and its APIs are defined and implemented in value.h and value.c.

The `value` struct wraps the underlying data and manages its memory according to the type users specified. In some cases the data is stored inline, other times it stores a pointer to the actual memory. Most of the time the `value` struct is transparent to users. The most common use case would be that user put the data into some data structure such as a deque. The deque will internally wrap the data into a value object. When user reads the data, the deque will unwrap the value and return the internal data. An exception here is when you work with iterators, the iterator will return the value wrapper directly, which tells you the type of the value data. This allows you to store different types of value data into one data structure without worrying about losing the type info of the data when iterating over the structure.

When you free the deque, deque will automatically free up all the data stored within. In other words, you won't ever need to iterate over the deque and free all the stored data manually and explicitly.

The `value` struct can also print out the wrapped data according to its type. This is convenient for debugging and building output – since `deque`, `ART` and `JSON` are also value types, and their internal data are all wrapped in `value`, their content can be easily printed out.

12.1.1 Types

We support the following value types:

type	type enum	free behavior (no-op if left blank)
`int8_t`	`ValueInt8`	
`uint8_t`	`ValueUInt8`	
`int16_t`	`ValueInt16`	

type	type enum	free behavior (no-op if left blank)
uint16_t	ValueUInt16	
int32_t	ValueInt32	
uint32_t	ValueUInt32	
int64_t	ValueInt64	
uint64_t	ValueUInt64	
char	ValueChar	
bool	ValueBool	
char*	ValueString	free()
char*	ValueStringR	
float	ValueFloat	
double	ValueDouble	
char*	ValueBASE64	free()
char*	ValueBASE64R	
struct json*	ValueJSON	pgmoneta_json_destroy(), this will recursively destroy internal data
struct json*	ValueJSONRef	
struct deque*	ValueDeque	pgmoneta_deque_destroy(), this will recursively destroy internal data
struct deque*	ValueDequeRe	

type	type enum	free behavior (no-op if left blank)
`struct art *`	`ValueART`	`pgmoneta_art_destroy()`, this will recursively destroy internal data
`struct art *`	`ValueARTRef`	
`void*`	`ValueRef`	
`void*`	`ValueMem`	`free()`

You may have noticed that some types have corresponding `Ref` types. This is especially handy when you try to share data among multiple data structures – only one of them should be in charge of freeing up the value. The rest should only take the corresponding reference type to avoid double free.

There are cases where you try to put a pointer into the core data structure, but it's not any of the predefined types. In such cases, we offer a few options:

- If you want to free the pointed memory yourself, or it doesn't need to be freed, use `ValueRef`.
- If you just need to invoke a simple `free()`, use `ValueMem`.
- If you need to customize how to destroy the value, we offer you APIs to configure the behavior yourself, which will be illustrated below.

Note that the system does not enforce any kind of borrow checks or lifetime validation. It is still the programmers' responsibility to use the system correctly and ensure memory safe. But hopefully the system will make the burden a little lighter.

12.1.2 APIs

12.1.2.1 pgmoneta_value_create Create a value to wrap your data. Internally the value use a `uintptr_t` to hold your data in place or use it to represent a pointer, so simply cast your data into `uintptr_t` before passing it into the function (one exception is when you try to put in float or double, which requires extra work, see pgmoneta_value_from_float/pgmoneta_value_from_double for details). For `ValueString` or `ValueBASE64`, the value **makes a copy** of your string data. So if your string is malloced on heap, you still need to free it since what the value holds is a copy.

```
pgmoneta_value_create(ValueString, (uintptr_t)str, &val);
// free the string if it's stored on heap
free(str);
```

12.1.2.2 pgmoneta_value_create_with_config

Create a value wrapper with `ValueRef` type and customized destroy and to-string callback. If you want to leave a callback as default, set the field to NULL.

You normally don't have to create a value yourself, but you will indirectly invoke it when you try to put data into a deque or ART with a customized configuration.

The callback definition is

```
typedef void (*data_destroy_cb)(uintptr_t data);
typedef char* (*data_to_string_cb)(uintptr_t data, int32_t format, char*
    tag, int indent);
```

12.1.2.3 pgmoneta_value_to_string

This invokes the internal to-string callback and prints out the wrapped data content. You don't usually need to call this function yourself, as the nested core data structures will invoke this for you on each of its stored value.

For core data structure types, such as `deque`, `ART` or `JSON`, there are multiple supported types of format: * `FORMAT_JSON`: This prints the wrapped data in JSON format * `FORMAT_TEXT`: This prints the wrapped data in YAML-like format * `FORMAT_JSON_COMPACT`: This prints the wrapped data in JSON format, with all whitespaces omitted

Note that the format may also affect primitive types. For example, a string will be enclosed by `"` in JSON format, while in TEXT format, it will be printed as-is.

For `ValueMem` and `ValueRef`, the pointer to the memory will be printed.

12.1.2.4 pgmoneta_value_data

Reader function to unwrap the data from the value wrapper. This is especially handy when you fetched the value with wrapper from the iterator.

12.1.2.5 pgmoneta_value_destroy

Destroy a value, this invokes the destroy callback to destroy the wrapped data.

12.1.2.6 pgmoneta_value_to_float/pgmoneta_value_to_double

Use the corresponding function to cast the raw data into the float or double you had wrapped inside the value.

Float and double types are stored in place inside the `uintptr_t` data field. But since C cannot automatically cast a `uintptr_t` to float or double correctly, – it doesn't interpret the bit representation as-is – we have to resort to some union magic to enforce the casting.

12.1.2.7 pgmoneta_value_from_float/pgmoneta_value_from_double For the same reason mentioned above, use the corresponding function to cast the float or double you try to put inside the value wrapper to raw data.

```
pgmoneta_value_create(ValueFloat, pgmoneta_value_from_float(float_val), &
    val);
```

12.1.2.8 pgmoneta_value_to_ref Return the corresponding reference type. Input `ValueJSON` will give you `ValueJSONRef`. For in-place types such as `ValueInt8`, or if the type is already the reference type, the same type will be returned.

12.2 Deque

The deque is defined and implemented in deque.h and deque.c. The deque is built upon the value system, so it can automatically destroy the internal items when it gets destroyed.

You can specify an optional tag for each deque node, so that you can sort of use it as a key-value map. However, since the introduction of ART and json, this isn't the recommended usage anymore.

12.2.1 APIs

12.2.1.1 pgmoneta_deque_create Create a deque. If thread safe is set, a global read/write lock will be acquired before you try to write to deque or read it. The deque should still be used with cautious even with thread safe enabled – it does not guard against the value you have read out. So if you had stored a pointer, deque will not protect the pointed memory from being modified by another thread.

12.2.1.2 pgmoneta_deque_add Add a value to the deque's tail. You need to cast the value to `uintptr_t` since it creates a value wrapper underneath. Again, for float and double you need to use the corresponding type casting function (`pgmoneta_value_from_float` / `pgmoneta_value_from_double`). The function acquires write lock if thread safe is enabled.

The time complexity for adding a node is O(1).

12.2.1.3 pgmoneta_deque_add_with_config Add data with type `ValueRef` and customized to-string/destroy callback into the deque. The function acquires write lock if thread safe is enabled.

```
static void
rfile_destroy_cb(uintptr_t data)
{
    pgmoneta_rfile_destroy((struct rfile*) data);
}

static void
add_rfile()
{
struct deque* sources;
struct rfile* latest_source;
struct value_config rfile_config = {.destroy_data = rfile_destroy_cb, .
    to_string = NULL};
...

pgmoneta_deque_add_with_config(sources, NULL, (uintptr_t)latest_source, &
    rfile_config);
}
```

12.2.1.4 pgmoneta_deque_poll Retrieve value and remove the node from the deque’s head. If the node has tag, you can optionally read it out. The function transfers value ownership, so you will be responsible to free the value if it was copied into the node when you put it in. The function acquires read lock if thread safe is enabled.

The time complexity for polling a node is O(1).

```
pgmoneta_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* tag = NULL;
char* value = (char*)pgmoneta_deque_poll(deque, &tag);

printf("%s, %s!\n", tag, value) // "Hello, world!"

// remember to free them!
free(tag);
free(value);
```

```
// if you don’t care about tag
pgmoneta_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* value = (char*)pgmoneta_deque_poll(deque, NULL);

printf("%s!\n", value) // "world!"

// remember to free it!
free(value);
```

12.2.1.5 pgmoneta_deque_poll_last Retrieve value and remove the node from the deque’s tail. If the node has tag, you can optionally read it out. The function transfers value ownership, so you will be responsible to free the value if it was copied into the node when you put it in. The function acquires read lock if thread safe is enabled.

The time complexity for polling a node is O(1).

```
pgmoneta_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* tag = NULL;
char* value = (char*)pgmoneta_deque_poll_last(deque, &tag);

printf("%s, %s!\n", tag, value) // "Hello, world!"

// remember to free them!
free(tag);
free(value);
```

```
// if you don't care about tag
pgmoneta_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* value = (char*)pgmoneta_deque_poll_last(deque, NULL);

printf("%s!\n", value) // "world!"

// remember to free it!
free(value);
```

12.2.1.6 pgmoneta_deque_peek Retrieve value without removing the node from deque's head. The function acquires read lock if thread safe is enabled.

The time complexity for peeking a node is O(1).

12.2.1.7 pgmoneta_deque_peek_last Retrieve value without removing the node from deque's tail. The function acquires read lock if thread safe is enabled.

The time complexity for peeking a node is O(1).

12.2.1.8 pgmoneta_deque_iterator_create Create a deque iterator, note that iterator is **NOT** thread safe

12.2.1.9 pgmoneta_deque_iterator_destroy Destroy a deque iterator

12.2.1.10 pgmoneta_deque_iterator_next Advance the iterator to the next value. You will need to call it before reading the first item. The function is a no-op if it reaches the end and will return false.

12.2.1.11 pgmoneta_deque_iterator_has_next Check if iterator has next value without advancing it.

12.2.1.12 pgmoneta_deque_iterator_remove Remove the current node the iterator is pointing to. Then the iterator will fall back to the previous node.

For example, for a deque `a -> b -> c`, after removing node `b`, iterator will point to `a`, then calling `pgmoneta_deque_iterator_next` will advance the iterator to `c`. If node `a` is removed instead, iterator will point to the internal dummy head node.

```c
// remove nodes without a tag
pgmoneta_deque_iterator_create(deque, &iter);
while (pgmoneta_deque_iterator_next(iter)) {
    if (iter->tag == NULL) {
        pgmoneta_deque_iterator_remove(iter);
    }
    else {
        printf("%s: %s\n", iter->tag, (char*)pgmoneta_value_data(iter->
           value));
    }
}
pgmoneta_deque_iterator_destroy(iter);
```

12.2.1.13 pgmoneta_deque_size Get the current deque size, the function acquires the read lock

12.2.1.14 pgmoneta_deque_empty Check if the deque is empty

12.2.1.15 pgmoneta_deque_to_string Convert the deque to string of the specified format.

12.2.1.16 pgmoneta_deque_list Log the deque content in logs. This only works in TRACE log level.

12.2.1.17 pgmoneta_deque_sort Merge sort the deque. The time complexity is O(log(n)).

12.2.1.18 pgmoneta_deque_get Get the data with a specific tag from the deque.

The time complexity for getting a node is O(n).

12.2.1.19 pgmoneta_deque_exists Check if a tag exists in deque.

12.2.1.20 pgmoneta_deque_remove Remove all the nodes in the deque that have the given tag.

12.2.1.21 pgmoneta_deque_clear

Remove all the nodes in the deque.

12.2.1.22 pgmoneta_deque_set_thread_safe

Set the deque to be thread safe.

12.3 Adaptive Radix Tree (ART)

ART shares similar ideas as trie. But it is very space efficient by adopting techniques such as adaptive node size, path compression and lazy expansion. The time complexity of inserting, deleting or searching a key in an ART is always O(k) where the k is the length of the key. And since most of the time our key type is string, ART can be used as **an ideal key-value map** with much less space overhead than hashmap.

ART is defined and implemented in art.h and art.c.

12.3.1 APIs

12.3.1.1 pgmoneta_art_create

Create an adaptive radix tree

12.3.1.2 pgmoneta_art_insert

Insert a key value pair into the ART. Likewise, the ART tree wraps the data in value internally. So you need to cast the value to `uintptr_t`. If the key already exists, the previous value will be destroyed and replaced by the new value.

12.3.1.3 pgmoneta_art_insert_with_config

Insert a key value pair with a customized configuration. The idea and usage is identical to `pgmoneta_deque_add_with_config`.

12.3.1.4 pgmoneta_art_contains_key

Check if a key exists in ART.

12.3.1.5 pgmoneta_art_search

Search a value inside the ART by its key. The ART unwraps the value and return the raw data. If key is not found, it returns 0. So if you need to tell whether it returns a zero value or the key does not exist, use `pgmoneta_art_contains_key`.

12.3.1.6 pgmoneta_art_delete

Delete a key from ART. Note that the function returns success(i.e. 0) even if the key does not exist.

12.3.1.7 pgmoneta_art_clear

Removes all the key value pairs in the ART tree.

12.3.1.8 pgmoneta_art_to_string Convert an ART to string. The function uses an internal iterator function which iterates the tree using DFS. So unlike the iterator, this traverses and prints out keys by lexicographical order.

12.3.1.9 pgmoneta_art_destroy Destroy an ART.

12.3.1.10 pgmoneta_art_iterator_create Create an ART iterator, the iterator iterates the tree using BFS, which means it won't traverse the keys by lexicographical order.

12.3.1.11 pgmoneta_art_iterator_destroy Destroy an ART iterator. This will recursively destroy all of its key value entries.

12.3.1.12 pgmoneta_art_iterator_remove Remove the key value pair the iterator points to. Note that currently the function just invokes `pgmoneta_art_delete()` with the current key. Since there's no rebalance mechanism in ART, it shouldn't affect the subsequent iteration. But still use with caution, as this is not thoroughly tested.

12.3.1.13 pgmoneta_art_iterator_next Advance an ART iterator. You need to call this function before inspecting the first entry. If there are no more entries, the function is a no-op and will return false.

12.3.1.14 pgmoneta_art_iterator_has_next Check if the iterator has next value without advancing it.

```
pgmoneta_art_iterator_create(t, &iter);
while (pgmoneta_art_iterator_next(iter)) {
    printf("%s: %s\n", iter->key, (char*)pgmoneta_value_data(iter->value))
        ;
}
pgmoneta_art_iterator_destroy(iter);
```

12.4 JSON

JSON is essentially built upon deque and ART. Find its definition and implementation in json.h and json.c.

Note that this document will not cover the iterative parsing APIs, since those are still experimental.

12.4.1 APIs

12.4.1.1 pgmoneta_json_create Create a JSON object. Note that the json could be an array (`JSONArray`) or key value pairs (`JSONItem`). We don't specify the JSON type on creation. The json object will decide by itself based on the subsequent API invocation.

12.4.1.2 pgmoneta_json_destroy Destroy a JSON object

12.4.1.3 pgmoneta_json_put Put a key value pair into the json object. This function invokes `pgmoneta_art_insert` underneath so it will override the old value if key already exists. Also when invoked for the first time, the function sets the JSON object to `JSONItem`, which will reject `pgmoneta_json_append` from then on. Note that unlike ART, JSON only takes certain types of value. See JSON introduction for details.

12.4.1.4 pgmoneta_json_append Append a value entry to the json object. When invoked for the first time, the function sets the JSON object to `JSONArray`, which will reject `pgmoneta_json_put` from then on.

12.4.1.5 pgmoneta_json_remove Remove a key and destroy the associated value within the json item. If the key does not exist or the json object is an array, the function will be no-op. If the JSON item becomes empty after removal, it will fall back to undefined status, and you can turn it into an array by appending entries to it.

12.4.1.6 pgmoneta_json_clear For `JSONArray`, the function removes all entries. For `JSONItem`, the funtion removes all key value pairs. The JSON object will fall back to undefined status.

12.4.1.7 pgmoneta_json_get Get and unwrap the value data from a JSON item. If the JSON object is an array, the function returns 0.

12.4.1.8 pgmoneta_json_contains_key Check if the JSON item contains a specific key. It always returns false if the object is an array.

12.4.1.9 pgmoneta_json_array_length Get the length of a JSON array

12.4.1.10 pgmoneta_json_iterator_create Create a JSON iterator. For JSON array, it creates an internal deque iterator. For JSON item, it creates an internal ART iterator. You can read the value or the array entry from `value` field. And the `key` field is ignored when the object is an array.

12.4.1.11 pgmoneta_json_iterator_next Advance to the next entry or key value pairs. You need to call this before accessing the first entry or kv pair.

12.4.1.12 pgmoneta_json_iterator_has_next Check if the object has the next entry or key value pair.

12.4.1.13 pgmoneta_json_iterator_destroy Destroy the JSON iterator.

12.4.1.14 pgmoneta_json_parse_string Parse a JSON string into a JSON object.

12.4.1.15 pgmoneta_json_clone Clone a JSON object. This works by converting the object to string and parse it back to another object. So the value type could be a little different. For example, an `int8` value will be parsed into an `int64` value.

12.4.1.16 pgmoneta_json_to_string Convert the JSON object to string.

12.4.1.17 pgmoneta_json_print A convenient wrapper to quickly print out the JSON object.

12.4.1.18 pgmoneta_json_read_file Read the JSON file and parse it into the JSON object.

12.4.1.19 pgmoneta_json_write_file Convert the JSON to string and write it to a JSON file.

13 Troubleshooting

13.1 Could not get version for server

If you get this FATAL during startup check your PostgreSQL logins

```
psql postgres
```

and

```
psql -U repl postgres
```

And, check the PostgreSQL logs for any error.

Setting log_level to DEBUG5 in pgmoneta.conf could provide more information about the error.

14 Acknowledgement

14.1 Authors

pgmoneta was created by the following authors:

```
Jesper Pedersen <jesperpedersen.db@gmail.com>
David Fetter <david@fetter.org>
Will Leinweber <will@bitfission.com>
Luca Ferrari <fluca1978@gmail.com>
Nikita Bugrovsky <nbugrovs@redhat.com>
Mariam Fahmy <mariamfahmy66@gmail.com>
Jichen Xu <kyokitisin@gmail.com>
Saurav Pal <resyfer.dev@gmail.com>
Bokket <bokkett@gmail.com>
Haoran Zhang <andrewzhr9911@gmail.com>
Hazem Alrawi <hazemalrawi7@gmail.com>
Shahryar Soltanpour <shahryar.soltanpour@gmail.com>
Shikhar Soni <shikharish05@gmail.com>
Nguyen Cong Nhat Le <lenguyencongnhat2001@gmail.com>
Chao Gu <chadraven369@gmail.com>
Luchen Zhao <lucian.zlc@gmail.com>
Joan Jeremiah J <joanjeremiah04@gmail.com>
Iury Santos <iuryroberto@gmail.com>
Palak Chaturvedi <palakchaturvedi2843@gmail.com>
Jakub Jirutka <jakub@jirutka.cz>
Mario Rodas
Annupamaa <annu242005@gmail.com>
Ashutosh Sharma <ash2003sharma@gmail.com>
Mohab Yaser <mohabyaserofficial2003@gmail.com>
Georg Pfuetzenreuter <mail@georg-pfuetzenreuter.net>
Ahmed Ashour <a8087027@gmail.com>
Sangkeun J.C. Kim <jchrys@me.com>
Tejas Tyagi <tejastyagi.tt@gmail.com>
Aryan Arora <aryanarora.w1@gmail.com>
Arshdeep Singh <balarsh535@gmail.com>
Din Xin Chen <s990346@gmail.com>
Mingzhuo Yin <yinmingzhuo@gmail.com>
Vanes Angelo <k124k3n@gmail.com>
Bassam Adnan <mailbassam@gmail.com>
```

14.2 Committers

```
Jesper Pedersen <jesperpedersen.db@gmail.com>
Haoran Zhang <andrewzhr9911@gmail.com>
```

14.3 Contributing

Contributions to **pgmoneta** are managed on GitHub

- Ask a question
- Raise an issue
- Feature request
- Code submission

Contributions are most welcome!

Please, consult our Code of Conduct policies for interacting in our community.

Consider giving the project a star on GitHub if you find it useful. And, feel free to follow the project on X as well.

15 License

```
Copyright (C) 2025 The pgmoneta community

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following
disclaimer in the documentation and/or other materials provided
with the distribution.

3. Neither the name of the copyright holder nor the names of
its contributors may be used to endorse or promote products
derived from this software without specific prior written
permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
```

BSD-3-Clause

15.1 libart

Our adaptive radix tree (ART) implementation is based on The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases and libart which has a 3-BSD license as

```
Copyright (c) 2012, Armon Dadgar
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

```
* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.

* Neither the name of the organization nor the
  names of its contributors may be used to endorse or promote products
  derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ARMON DADGAR
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```