Zen Embedded Database for Nemerle 0.1 (Zdb)

Zdb предназначена для замены стандартной сериализации в следующих случаях:

- объем сохраняемых данных составляет десятки, сотни мегабайт;
- данные представляют собой сложно организованные структуры (сети, графы), алгоритм навигации по ним плохо реализуем (сложный, медленный) с реляционными и пост-реляционными СУБД.
- извлечения из базы носят множественный атомарный характер, определяемый итерационным алгоритмом.
- скорость доступа имеет решающее значение.
- данные имеют иерархическую организацию и могут располагаться в нескольких связанных хранилищах.

Возможности Zdb:

- ленивая загрузка объектов “по требованию”.
- разделяемые данные: объекты из одной базы данных могут ссылаться на объекты, расположенные в другой.
-ленивая загрузка связанных баз данных - загрузка базы с интересующим объектом “по требованию”.
-автоматизм ленивых загрузок и сохранений.
-поддержка стандартных и нестандартных контейнеров, в том числе Nemerle.NList.
-поддержка вариантов.
-транзакции и автотранзакции. Последнее означает, что Commit производится автоматически при закрытии базы, если до этого не была команда Cancel.

Ограничения Zdb:

-максимальное число связанных баз данных (кластера) - 15. для 32-бит Windows.
-максимальный объем одной базы данных - 1 гигабайт. для 32-бит Windows.
-только .Net 4.0
-Persist классы не могут быть generic.
-Debug Config dll только. - временно.

Использование.

Для использования Zdb в своем проекте добавьте ссылку на DBLib.dll, в исходном коде откройте пространство имен DBLib.

Описание сохраняемых (Persist) классов:

```
class Px
  ZT str : string
  ZT contains : List.[Px]
  Zs unsup : Dictionary.[string, object]
  private cnt : int
```

При описании сохраняемых полей класса используются два определения:

ZT - (оптимально по скорости). Применима к Persist классам, строкам, value - типам, их контейнерам. Поле трансформируется в public свойство с таким именем.

Zt - То же, но поле трансформируется в private свойство с таким именем.

ZS - для всех остальных типов используется стандартная .Net сериализация,
поэтому тип поля должен иметь интерфейс ISerializable.
Поле трансформируется в public свойство с таким именем.
Zs - то же, но поле трансформируется в private свойство с таким именем.

В вышеописанном классе Px имеется три сохраняемых поля и одно временное - cnt, ограниченное в существовании временем жизни объекта в оперативке. Описанные таким образом объекты становятся сохраняемыми, для них генерируется код поддержки работы с Zdb и поддержка интерфейса IPersist, предоставляющего набор методов для работы с объектом, среди которых свойство Dbase, позволяющее установить базу данных объекта при его создании.

Открытие/создание базы

xbase = Zdb("filename1"[, размер]) //база данных может содержать ссылки на внешние объекты, размер по умолчанию или указывается.

ZdbClear("filename2"[, размер]) //то же, но база открывается как пустая. Данные стираются, если были. До вызова Commit() состояние на диске не изменяется.

Zdb1("filename2"[, размер]) //база данных не может содержать ссылки на внешние объекты, размер базы данных устанавливается в 1 << 25, без этого параметра увеличивается по мере необходимости.

Zdbi("filename3"[, размер]) //база данных может содержать ссылки на внешние объекты, и все связанные базы должны быть сразу открыты, т. е. добавка 'i'mmediately - отмена ленивости загрузки для внешних баз.

Сохранение Root объекта
Это необходимое действие при инициализации базы данных.

```
def root = Px() <- {str = "root object"; contains = List()}

xbase.Root = root
```

Zdb - чистая объектная баз данных, root объект - это корневой объект, который прямо или опосредованно содержит ссылки на все объекты в базе.
Если открытая база данных содержит записанные данные, то вначале получаем root объект

```
def root = z.Root :> Px
```

и дальше работаем как с обычным объектом, Zdb будет самостоятельно разрешать ссылки на объекты, доставая их из базы и создавая в оперативной памяти по мере необходимости.

```
def rcount = root.contains?.Count
```

Создание и добавление других объектов

При создании объекта можно не указывать явно его базу данных, ею автоматически считается последняя открытая

```
root.contains.Add(Px())
```

Если объект принадлежит другой базе, то можно указать ее при создании

```
s = Px(zbase)
```

Объект не может изменить свою базу данных, попытка этого вызывает исключение.

Объект из одной базы данных может ссылаться на объекты из другой, если его база данных объекта открыта как **Zdb** или **Zdbi.**

```
s = Px(zbase)
d = Px(ybase)
s.contains = List([d])
```

При изменении не-персистных объектов, **сохраненных в базе**, к которым относятся стандартные контейнеры и все прочие не value и string объекты, не поддерживающие интерфейс IPersist, необходимо явно указать Zdb требование их сохранения. Это обусловлено тем, что их изменение, в отличие от Persist объектов, не отслеживается Zdb, а сравнение при каждой транзакции содержимого всех таких объектов с текущим состоянием в оперативной памяти хоть и возможно, но нецелесообразно. Если на не-персистный объект ссылаются персистные объекты из разных баз, то каждая база сохранит его копию. Это сделано для предотвращения образования неожиданных связей между базами.

Команда **put** оповещает Zdb о необходимости обновления некоторого неперсистного объекта через содержащий его персистный объект.

```
put root.contains
```

Присвоения полям персистных объектов строк, value и персистных объектов Zdb отслеживает и фиксирует изменения автоматически , никаких дополнительных команд не требуется.

Персистные варианты.

```
[Zv] public variant Var
  | A
    r : string
    s : double
```

```
 | B
   z : int
   next : Var
```

Для эффективной работы с вариантами в Zdb необходимо добавить атрибут Zv к объявлению. Без этого работа с вариантом будет строится как с неизвестным объектом, использующим стандартную .Net сериализацию.

```
class Y : Px
  ZT var1 : Var
  ZT cv  : Dictionary.[Var, Var]
  ZT lv  : list.[Var]
```

Определение Y будет корректно скомпилировано, если Var будет объявлен с атрибутом **Zv**, иначе будут выданы три предупреждения о невозможности использования быстрого сохранения и применении вместо него сериализации .Net.

Транзакции.

Состояние БД на диске не изменяется до вызова Zdb.Commit(). Отмена записи - Cancel(). Если потребовалось отменить изменения, лучше всего завершить работу с БД, т. к. Cancel отменяет сохранения в буфере сохранений, но не изменения объектов в памяти. **Важно.** При закрытии базы данных Commit происходит автоматически. Если буфер сохранения чист или очищен вызовом Cancel, Commit соответственно ничего не записывает.