Table of Contents

1. Release Notes for Statice 2.1

This section discusses improvements offered by Statice 2.1 and some known limitations of Statice 2.1.

1.1. Statice Improvements for Genera 8.1

- On UX400 systems, the Sun no longer times out on some network connections with large transactions.
- Using the **number** type should no longer generate errors.
- Statice works more reliably on 3600-family machines.
- Problems relating to secondary mapping errors have been corrected.
- The deadlock detector no longer aborts non-deadlocked transactions. This should help avoid the `Too many retries` error, especially if a process holding locks is in the debugger.
- Aborts are now more fair. This should help avoid livelocks.
- Problems with background invalidation have been corrected. This fix, plus some others should dramatically improve multi-user performance.

1.2. Statice and CLOS

Genera 8.1 includes CLOS (Common Lisp Object System). Statice users might be interested in how Statice and CLOS interact. In general, there is no direct integration between Statice 2.1 and Genera 8.1. However, using Statice and CLOS in the same Lisp world works, and you can develop programs that use both Statice and CLOS.

In Genera 8.1 and Statice 2.1, Statice entity handles are implemented as Flavors instances, not CLOS instances. Statice attribute accessing functions are Flavors generic functions, not CLOS generic functions. You cannot define methods for user-defined CLOS generic functions that are specialized on a flavor (such as a Statice entity flavor).

Statice users can use CLOS, noting the following restrictions. You cannot mix CLOS classes into a Statice entity definition. See the section "Mixing Flavors Into a Statice Entity Definition" in *Statice*. You cannot define CLOS classes that inherit from Statice entity flavors, or define CLOS methods that specialize on Statice entity flavors. We anticipate that some of these restrictions will be lifted in a future release of Genera.

1.3. Limitations on statice:for-each* in Statice 2.1

In Statice 2.1, **statice:for-each*** is restricted to single-variable queries. Queries involving more than one variable are supported by **statice:for-each**, but not by **statice:for-each***. See the section "Using **statice:for-each** on Many Variables" in *Statice*.

Also, **statice:for-each*** is restricted to iterating over all entities of an entity type; it cannot iterate over members of a set-valued attribute as **statice:for-each** can. See the section "Iterating Over Sets with **statice:for-each**" in *Statice*.

1.4. Duplicate Attribute Names Are Not Allowed in Statice 2.1

Statice does not allow an entity to specify an attribute name which is the same as an attribute name of one of its parent entity types.

This restriction is one difference between Statice and Flavors (and CLOS). A flavor can specify the name of an instance variable even if it is the name of an inherited instance variable, with the goal of modifying it by giving it a new option, or an option to override an inherited one.

1.5. Hints on Using Statice

- Use small transactions. If the log is not big enough, creating large databases with one large transaction takes longer than many short transactions. Use small transactions or specify a log size of double the intended database size when you build the file system. In any case, you need a log file that is at least as large as the number of pages created or written in any given transaction. Use the function **dbfs:show-dbfs-meters** on the server to look at the performance of Statice's transaction system. If there are many reinserted pages and/or the log has grown many times, you are probably using transactions that are too long, with logs that are too small. Long transactions also thwart the buffer replacement mechanism, which causes paging overhead.

- Do not use Update Database Schema. There are currently some reliability problems with Update Database Schema. The best way to change a schema is to build a new database with the new schema. If you do decide to use Update Database Schema, make a backup of your database before you start.

- Use TCP protocols for `DBFS-PAGE` and `ASYNC-DBFS-PAGE` namespace services. Do not use chaos protocols. Remove all chaos protocols for `DBFS-PAGE` and `ASYNC-DBFS-PAGE` from the namespace objects for your hosts. Rebuild worlds with the new namespace information at your convenience. Make sure all your hosts have `INTERNET` addresses.

- Do not warm boot. Warm booting machines currently does not work well if Statice locks are held. If you have to warm boot, just do so long enough to save your buffers. Do not use Statice after a warm boot.

- Do not override **sys:without-aborts**. If Statice asks you not to use `c-ABORT`, heed its advice. If the machine is stuck, it is better to just boot the machine rather than forcibly abort critical Statice functions.

- If lookup or deletion is slow, use indexes. If accessing set-valued attributes is slow then indexes on sets may be very helpful, even if each entity only has a few items in its set valued attribute. This is especially the case if you have a lot of those entities. Indexes (especially inverse indexes) will help speed up deletion. Deletion must preserve the integrity of the database, by removing all references to the object you are deleting. If there are inverse indexes which can help find those references, then deleting is very fast. If there are indexes missing, then a scan of the database is necessary, which can be very slow for large databases. Avoid the use of type **t**, when possible. Use **user::all-but-entity** instead, if you can.

- Be careful about side-effects inside the dynamic scope of a **user::with-transaction**. Transactions can abort for many reasons. When they are aborted, they are automatically restarted, including the user code inside the dynamic scope of the **user::with-transaction** form. Be careful about modifying variables, flavor instances, and structures which are created before the transaction, and which will exist after it.

- ```
  ;;Wrong
  (let ((a nil))
    (with-transaction ()
        ;;Any code which looks at A here could have
        ;;problems if the transaction is aborted.
        ...
      (setq a (make-person :name "Frank"))
      ...))
  ;;Right
  (with-transaction ()
     ...
     (let ((a (make-person :name "Frank")))
       ...))
  ```

- Avoid nested transactions. If the function **dbfs:show-dbfs-meters** on your client machine shows many nested transactions, then you may want to rethink the modularity of your software somewhat. It is best not to nest transactions, since the current page state is saved before the inner transaction is executed. If you have modified 1000 pages before doing a nested transaction, then 1000 pages have to be copied to another buffer. This copying not only takes a lot of time, it also hurts your paging performance. It is most efficient to just have one with-transaction in effect at any given time.
  ```

- Avoid the use of type **number**, when possible. It is slow, and uses more storage than a specific type of number, such as **single-float** or **fixnum**.

- Save worlds with Statice, do not just load Statice every time you want to use it. This avoids a lengthy delay everytime you boot, and it is a more reliable way to use Statice. Be sure to observe the following:

  ° Build your worlds correctly. Always start a world building procedure by booting the latest released world from Symbolics. Disable services. Load the IP-TCP system first (IP-TCP is a loadable system bundled with Genera 8.1), then load Statice. This insures that Statice knows it can use IP-TCP.

  ° Do not save a Statice world without IP-TCP, since Statice is not reliable over Chaos.

  ° Be sure any critical private patches are loaded.

  ° Save the resulting world, then immediately reboot the new world. Never use a world after you have just saved it.

  For more information, see the section "Making, Distributing, and Using Worlds" in *Site Operations*.

- It is better to keep all your schema for a particular database in one file, and recompile the entire schema file when you make schema changes.

- Use unique attribute names. Do not use the same name for two different attributes in two different object definitions. This is a current limitation.

- The **limited string** type seems to be using a non-compact form of storage. Its use is not recommended until this bug can be fixed.

Index