Customizable and User-Defined Stored Procedures

1. The SQL content of LeasePak Databases

LeasePak databases contain, in addition to tables and indexes, a large number of SQL objects, which generically we refer to as "Stored Procedures" or "SQL Code". There are a number of different kinds of SQL Code Objects, each characterized by a specific naming convention, origin and purpose.

---

2. Kinds of SQL Code Objects

SQL Code Object Type	Object Naming Convention	File Naming Convention
McCue Triggers	mt_*	<dbms>_mt_*.sql
McCue Procedures	mp_*	<dbms>_mp_*.sql
McCue Packages	mpkg_*	ora_mpkg_*.sql
Custom Procedures	cp_*	<dbms>_cp_*.sql
User Procedures	up_*	<dbms>_up_*.sql
User Packages	upkg_*	ora_upkg_*.sql

Note that Packages are a programming construct unique to Oracle and are not available in Sybase databases.

---

3. Origins and Maintenance of SQL Code Objects

The SQL Code Objects designated as "McCue" are originated and maintained exclusively by McCue Systems. Modification of any of these code objects by the end user is strongly discouraged and may render a site unsupportable. These objects are invoked by the LeasePak server and the LeasePak client and should generally not be invoked by the end user.

The SQL Code Objects designated as "Custom" are originated by McCue Systems. Their interfaces are defined and maintained by McCue Systems. Their functionality, as delivered to the end user, is intended to be a reasonable implementation of the purposes for which the procedures are intended, and does not require modification by the end user in order to utilize LeasePak. However, they are also intended as entry points into code that may be customized by the end user. Only their interfaces and calling conventions are specified by McCue Systems.

The SQL Code Objects designated as "User" are originated and maintained by the end user exclusively. The purpose of the "User" code objects is to allow the end user free reign in implementing the custom functionality of the "Custom" code objects. McCue Systems does not specify or deliver any "User" code objects; code originated by and delivered by McCue Systems will never reference "User" code objects.

---

4. Calling sequences

The SQL Code Objects are arranged in a specific hierarchy. This hierarchy controls which code objects can call other code objects. Both Sybase and Oracle require that called code objects be already defined in the logical database when they are first referenced by a calling code object. This necessitates a specific sequence for loading the code objects into a new or existing logical database.

Two mechanisms exist for determining this sequence. The first mechanism applies to SQL code objects delivered by McCue Systems as standard components. This mechanism applies to SQL code objects designated as "McCue" (mt's, mpkg's and mp's) and as "Custom" (cp's). This first mechanism operates by loading the four covered kinds of code objects in a specific order (mpkg's first, then mp's, and then mt's and finally, the cp's); within each kind of code object, the load order is strictly ASCII sort order by filename.

The second mechanism applies to SQL code objects originated by the end user; i.e., those designated as "User" code objects (upkg's and up's). This second mechanism operates by loading the upkg's and up's in the order specified by a file created and maintained by the end user; if the file does not exist at the time of load, then it is created by listing first the upkg's and then the up's, and then by ASCII sort order by filename within each kind of code object.

In either case, whether the file is created by the end user or by the system by default as described above, the file is used by the loader to determine the sequence in which the individual files are to be loaded. The "User" code object files may be listed in any order that suits the needs of the site's customized code; if the end user lists the code objects in an order that does not provide for predefinition of objects, then the code load will fail.

In any case, the load order is first the "User" code objects, according to the load sequence file, then the "McCue" code objects, and finally the "Custom" code objects. This arrangement prevents either "User" code objects or "McCue" code objects from calling "Custom" code objects.

If the end user's implementation of the functionality of a "User" code object requires that the "User" code object be able to call a "Custom" code object, then the end-user will have to refactor the functionality of the "Custom" code object into one or more "User" code objects and utilize the load sequence file mechanism to insure that the "User" code objects will be loaded in the desired order.

---

5. Content of SQL Code Object source files

It is required that each "Custom" or "User" source file declare exactly one SQL code object. That is, each file named <dbms>_cp_*.sql, <dbms>_upkg_*.sql or <dbms>_up_*.sql is allowed to define only one Custom Procedure, User Package or User Procedure, respectively. It is further required that each such code object defined by such files must bear the exact same name as the file, after removing the filename's

<dbms>_ prefix and the .sql file extension. In other words, a file named "ora_cp_altlsesum.sql" may define only one object, and that object must be named "cp_altlsesum". User Packages may define only one package, but each package may contain multiple functions.

The primary reason for this convention is to allow the loader to automatically, without user intervention or user code, perform the necessary grant of privileges to the logical database role or group to allow for the code object's execution.

The source files must also contain certain programming constructs designed to allow the system to manage the creation and replacement of code objects. The required constructs are given in Appendix A (Oracle) and Appendix B (Sybase) of this document. Without these constructs, the loading of SQL code objects will fail, if not on the first load, then on a subsequent one.

---

6. The SQL Code Object loading process

The loading process of SQL code objects occurs under 4 circumstances.

Logical Database Create: The first circumstance is on initial creation of a LeasePak logical database. During the creation process, performed by the script db_create, the script db_load_code is invoked. This is the script that implements the load order rules described above. Db_load_code also creates the script for granting privileges to the user code objects which is later executed by db_set_security, also invoked by db_create.

Logical Database Upgrade: The second circumstance is on upgrade of a LeasePak logical database to a new LeasePak release. During the upgrade process, the database conversion driver, sgenlpux_conversion, invokes the script db_drop_code, which enumerates and drops all code objects within the logical LeasePak database being upgraded. Subsequently, the conversion driver invokes db_load_code as described under "Logical Database Create" above.

Installation of Maintenance Build: The third circumstance is on installation of a Maintenance Build of LeasePak of the same release. Maintenance Builds may require replacement of certain code objects, as well as other modifications, in order for the logical LeasePak database to remain compatible with the newly delivered server or client components.

Development of Customized Code Objects: the fourth circumstance is during the user's development cycle for customizing LeasePak SQL code objects. When the end user is ready to test an iteration of his development, the "User" and "Custom" code objects may be replaced in a given database environment by invoking db_load_code and db_set_security with special options that restrict their functionality to the end-user's code.

---

7. Location of McCue Systems-provided and End-User Code

All McCue provided SQL code is delivered in each LeasePak build in the build's sql directory. Each supported DBMS has its own subdirectory under sql, such as sql/ora and sql/syb.

> LeasePak database environments, when first created, point via symbolic links to a particular build. Typically this build is the one designated as "live". The live build may be replaced from time to time by the installation of Maintenance Builds; environments constructed to point to the live build will then point to whichever build is currently designated as live. Within the environment, there are various "directories" which are really symbolic links to various components of the referenced build. In particular, there is always a "sql" link that links to the build's sql subdirectory for that environment's particular DBMS. Therefore, a Sybase environment's sql link points to the build's sql/syb directory, and an Oracle environment's sql link points to the build's sql/ora directory.

Regardless of the DBMS used by an environment's logical LeasePak database, the link is named "sql" and may always be referred to by the shell environment variable "$usql" from processes configured to point to that environment.

Upon installation of LeasePak version 60a, the installing user is asked to provide a Custom Code Directory. By default, this is /opt/msi/cst, though the user may locate this directory anywhere outside of the LeasePak release directory ($TOPDIR). This directory must already exist at installation time and must be writable by the MSIADMIN user. The location of this directory is stored in the environment variable "$CSTDIR".

Upon installation of each build, either through the initial installation or subsequent maintenance builds, will, upon being linked as the "live" build, be "brought online;" i.e., will be installed in the system in a fully usable way. This process is new to v60a, and is centered around the concept of user-customizable code. When the build is onlined, a subdirectory in $CSTDIR is created using the build's build sequence (e.g., $CSTDIR/6.00.1234) and a symbolic link in the build directory is created pointing to this directory. This link is named "cst". Therefore, every build will have a link "cst" pointing to the directory $CSTDIR/<buildseq>. Also at this time, further subdirectories in $CSTDIR/<buildseq> are created, "cql" and "prg"; cql will have a subdirectory for each installed DBMS in the release: "cql/syb and/or cql/ora". These directories are owned by $MSIADMIN and have 750 permissions.

It is into the cql subdirectories that the end-user must place all customized "custom" SQL code objects and all "user" SQL code objects. The load order file described above is also located here, and must be named "<dbms>_load_seq.txt". All files located in the cql subdirectories must be named using the above described naming conventions, and must be owned by ${MSIADMIN}:${MSIGROUP}, with 440 (-r--r-----) permissions.

The user should never customize the <dbms>_cp_*.sql files found in the build's sql subdirectories. Instead, the end user should make copies of these delivered "custom" SQL code object files in the appropriate cql subdirectory and perform their customizations there.

When a database environment is created or upgraded, an environment subdirectory (really a symbolic link) named "cql" will be created, pointing to the appropriate DBMS subdirectory within the $CSTDIR/<buildseq>/cql directory. This link may be referred to using the shell environment variable "$ucql" from any process configured to point to that database environment. Through a special option to the setup_new_env script, a database environment's $ucql may be made to point to $usql, effectively preventing any code object customizations from affecting the logical LeasePak database referenced by that particular database environment.

Likewise, each database environment will have another link named "prg", pointing to the $CSTDIR/<buildseq>/prg directory. This directory is intended for user customizable and user-provided programs and scripts which LeasePak will deliver or specify from time to time.

When db_load_code runs, it will gather lists of the file names for each of the defined SQL code object types. Using the list of <dbms>_cp_*.sql filenames, obtained exclusively from the McCue-delivered $usql directory, the loader will check for an instance of each such file in the $ucql directory. If the loader finds such a file in the $ucql directory, then it will replace the pathname in the list with the pathname of the $ucql version, allowing the end-user's customized code to override the standard McCue-delivered versions of the same code objects.

If $ucql is empty, no end-user code will be loaded into the database.

To summarize:

SQL Code Object Type	File Naming Convention	Default Location	Custom code Location
User-defined package	ora_upkg_*.sql	n/a	$ucql
User-defined procedure	<dbms>_up_*.sql	n/a	$ucql
Load sequence file	<dbms>_load_seq.txt	n/a	$ucql
McCue procedure	<dbms>_mp_*.sql	$usql	n/a
McCue package	ora_mpkg_*.sql	$usql	n/a
McCue trigger	<dbms>_mt_*.sql	$usql	n/a
Custom procedure	<dbms>_cp_*.sql	$usql	$ucql

8. Process of customizing SQL Code Objects

The following outlines the steps the end-user must perform in order to implement customized SQL code objects:

a. On initial installation of build (including initial installation of a LeasePak release):

(1) Determine which Custom Code Objects are to be customized;

(2) Copy those code object files to the appropriate cql dbms subdirectory within $CSTDIR;

(3) Perform desired customizations, including creation of User Code Objects, and a load sequence file if necessary;

(4) Unit test the customized code by:

(a) Dropping the existing end-user code objects from the targeted logical LeasePak database using db_drop_code (see Appendix C),

(b) Loading the customized code into the targeted logical LeasePak database using db_load_code (see Appendix C),

(c) Granting necessary privileges to the database users of the targeted logical LeasePak database using db_set_security (see Appendix C).

(5) Execute the pertinent LeasePak functionality within the targeted logical LeasePak database.

b. Once the code has been customized, new logical LeasePak databases may be built, or existing LeasePak database environments in prior releases may be upgraded, or existing LeasePak database environments in the current release may be updated as described above.

c. On installation of a new build into an existing LeasePak release:

(1) Determine if there have been any changes made to the interfaces (i.e., parameter lists or contents of returned result sets) or functionality or implementation details of any of the "Custom" procedures that the end-user previously has customized;

(2) Alter the customized copies of such code objects as have changed so that they are compatible with the new build;

(3) Unit test the customized code by loading the current version of the Custom and User code objects using db_load_code and then executing the pertinent LeasePak functionality.

Appendix A - Oracle package header

The following are the required programming constructs for an Oracle package file:

```
Whenever Sqlerror Exit Failure Rollback
Whenever Oserror  Exit Failure Rollback

CREATE OR REPLACE PACKAGE upkg_<package-name>
AS
      <procedure and function prototypes>
END upkg_<package-name>;

CREATE OR REPLACE PACKAGE BODY upkg_<package-name>
AS
      <procedure and function definitions>
END upkg_<package-name>;
\
Exit
```

and for an Oracle procedure file:

```
Whenever Sqlerror Exit Failure Rollback
Whenever Oserror  Exit Failure Rollback

CREATE OR REPLACE PROCEDURE cp_<procedure-name>
      <procedure definiton>
END cp_<procedure-name>;
\
Exit
```

Appendix B - Sybase procedure header

The following are the required programming constructs for a Sybase procedure file:

```
:r LeasePak_syb_use
go
if exists
(select * from sysobjects where name = 'up_<procedure-name>')
     drop procedure up_<procedure-name>
go
create procedure up_<procedure-name>
      <body of procedure>
go
```

Appendix C - new flags added to existing scripts

(1) setup_new_env

```
[test60a:/home/test60a] setup_new_env
ERROR: Arguments required:
  Usage: setup_new_env [-[dblncfts]] env-name db-type [db-server db-name] [build-
         descriptor]
  Usage: setup_new_env -v[dbln] env-name host-env build-descriptor
  where:
      -v = Visitor environment          -t = Test driver environment  [require build-
                                                                       descriptors]
      -n = No client info display       -f = Foreign (non-LeasePak) database
      -c = Closed to visitor            -l = Links used to populate exe dir
      -b = Build-id used for live/dlvy  -d = Dev startup files used
      -s = No end-user code
  A build descriptor is 'live', 'dlvy', 'host', or a build name ('bld#.##.####',
  e.g., 'bld5.01.1126')
```

The new -s flag will prevent any end-user code ("User" code objects or customized "Custom" code objects) from being loaded into logical LeasePak database.

(2) db_drop_code

```
[dba60a:/home/dba60a] db_drop_code
ERROR: Usage: db_drop_code [-u] environment-name [dbo-password]
```

The new -u flag will cause db_drop_code to drop only end-user code ("User" code objects or customized "Custom" code objects) from the logical LeasePak database indicated. In the absence of the –u flag, db_drop_code will drop all end-user and McCue-provided code including views from the logical LeasePak database indicated.

Note that, starting in v53a, db_drop_code also affects views within the logical LeasePak database.

(3) db_load_code

```
[dba60a:/home/dba60a] db_load_code
ERROR: Usage: db_load_code [-u] environment-name [dbo-password]
```

The new -u flag will cause db_load_code to load only end-user code ("User" code objects or customized "Custom" code objects) into the logical LeasePak database indicated. In the absence of the –u flag, db_load_code will load all end-user and McCue-provided code objects including LeasePak views into the logical LeasePak database indicated.

Note that, starting in v60a, db_load_code also affects views within the logical LeasePak database.

(4) db_set_security

```
[dba60a:/home/dba60a] db_set_security
ERROR: Usage: db_set_security [-u] environment-name [dbo-password]
```

The new -u flag will cause db_set_security to execute only the script created by db_load_code when it last loaded end-user code ("User" code objects or customized "Custom" code objects) into the logical LeasePak database indicated.