SoftLanding SYSTEMS
A Division of the UNICOM Group of Companies

UNICOM SYSTEMS, INC.

Supplement #17

84 Elm Street • Peterborough, NH 03458 USA
TEL (010)1-603-924-8818 • FAX (010)1-603-924-6348
Website: http://www.softlanding.com Email: techsupport@softlanding.com

MANAGING SQL OBJECTS WITH TURNOVER® FOR ISERIES V100

OVERVIEW

The iSeries has always had a database. Its name has evolved over time; currently, it is called the DB2 Universal Database for iSeries. This name reflects IBM's desire to move the iSeries database closer to DB2 industry standards. To do this, IBM is adding more features to the database, many of which are available only through Structured Query Language (SQL), instead of the traditional Data Definition Statements (DDS). For this reason, and for many others, many IT shops are rapidly learning to define their databases with SQL instead of DDS.

TURNOVER® for iSeries v100 can help you manage your database, whether you build it using SQL or DDS.

This document describes how to use TURNOVER® for iSeries v100 to manage these SQL objects:

- Tables
- Indexes
- Views
- Triggers
- Constraints
- Functions (TURNOVER® 5.3 and higher)
- SQL stored procedures (TURNOVER® 5.3 and higher)
- Externally stored procedures and functions (TURNOVER® 5.3 and higher)
- Ad-hoc SQL statements
- Database security.

Other SQL terms that we will use in this document are:

- Row (record)
- Column (field).

One basic concept is common to managing all SQL-related items: You type the SQL Data Definition Language statements (referred to as "DDL") into a source member, and create the object by executing the **RUNSQLSTM** over the source member. This is similar to typing DDS statements into a source member, and compiling it using the **CRTPF** or **CRTLF** commands to create the objects.

Avoid drawing too many similarities between the two creation processes, though. While DDS *is* compiled from source, it's better to think of an SQL object as something that's generated from a script. For example, if there is an error in a line of your DDS source, the compiler will not generate that object. With SQL, processing stops – but some source statements may have already been executed, and they cannot be un-done. This may leave you with unexpected – and undesirable – results. For example, the table might get created, but the constraints won't be applied.

Additionally, DDS, by definition, has a one-to-one relationship between the source and the object. In contrast, SQL can generate many objects from a single script.

About SQL collections

The groups of related tables, indexes and views in a database are called "collections." (Collections are also sometimes referred to as "schemas.") A database collection is usually created only once during the application setup process. (Chances are good that your database collections were created before you started using TURNOVER® for iSeries v100.)

On the iSeries, SQL collections are implemented as libraries. Tables are implemented as physical files in the collection library, and indexes and views as logical files. Also found in a database collection are the triggers, constraints, functions, and stored procedures, which are implemented as programs and used to manipulate the data in the tables, indexes, and views.

We're often asked if TURNOVER® for iSeries v100 supports SQL collections. Just as TURNOVER® for iSeries v100 has no specific support for managing libraries, there is no specific support for managing an SQL collection. However, it is possible with SQL to have one giant script that executes a series of SQL statements to create an entire database, sometimes even populating it with some records. When customers ask us about TURNOVER® for iSeries v100 support for "collections," we suspect that they are in fact referring to this giant script.

The answer is – Yes, and No. Yes, TURNOVER® for iSeries v100 *can* execute that script for you, as you will see later in the section entitled ***Managing Programs that require*** SQL Packages

If your company uses SQL packages, then you will encounter a situation where program objects that you are promoting need to have SQL packages created or re-created.

There are two ways to handle this situation: post-run commands, or an exit program.

Post-run command method

When a *PGM on your form needs to have an SQL package created or re-created during the form run, you can link a post-run command to the form line for your program that executes the CRTSQLPKG command appropriately. Once you have done this with your program, TURNOVER® for iSeries v100 will thereafter prompt for the correct post-run command any time it encounters that object on a subsequent form.

Exit program method

You can write an exit program to use with one of TURNOVER® for iSeries v100's form processing exit points (for example, Exit 20). Your exit program could sift through your form, identify lines eligible for CRTSQLPKG processing, and issue the command for each eligible line. This technique is best employed in situations where more automation is desired.

Executing Ad-Hoc SQL Statements on page 29. However, when a script is executed in this manner, TURNOVER® for iSeries v100 has no real knowledge of what the script is going to do. Under such circumstances, TURNOVER® for iSeries v100 can't provide the protections and audit trail that most TURNOVER® for iSeries v100 customers demand.

This document describes UNICOM's recommendations for using TURNOVER® for iSeries v100 to manage your SQL objects. Initially, you might have to rework some of your existing scripts, but the benefit is that those objects can then be managed as reliably and effectively as TURNOVER® for iSeries v100 has managed DDS-defined databases for many years.

Therefore, the first recommendation we make with regard to managing your SQL objects follows.

UNICOM Systems, Inc. Recommends

To maintain a manageable one-to-one source/object relationship, use one DDL source member to create one SQL object.

About SQL object names

Names for many types of SQL objects can be much longer than the ten-character iSeries object name limit. SQL objects actually have two names: the *SQL name* and the *system name*. The SQL name is used internally by SQL, and the system name is used by non-SQL interfaces, such as high-level language programs and OS/400 commands. The system name is the name of the OS/400 object that appears in your collection library.

When you generate an SQL object with a name that is longer than ten characters, the iSeries automatically assigns a corresponding ten-character system name. For example, table **CustSalePart** might be renamed "CustS00001." As you might imagine, the resulting database might be confusing and difficult to manage. Therefore, we make these recommendations about naming your SQL objects.

UNICOM Systems, Inc. recommends:

1. **Always** name the source member the same as the System Name (for example, **CstSalePrt**).

2. **For Tables or Views:** Add a line at the end of the source to tell the system what you want to use as the short system name. For example:

   ```
   Rename Table CustSalePart To System Name CstSalePrt
   ```

   The SQL name *must* be more than 10 characters; otherwise the rename statement will also rename the SQL name. Under these circumstances, you can use the SQL statement **CREATE ALIAS** to create an "alias" for the SQL name.

3. **For Indexes:**

   ```
   Rename Index CustomerbyTerritory To System Name CustTerr
   ```

4. **For SQL Stored Procedures, Externally Stored Procedures, and Functions:** Use the **Specific** clause. For example:

   ```
   Create Procedure MyLongProcedure Specific MyLongPr
   ```

5. **For Constraints:** You don't have to worry about these because the iSeries object isn't visible and the **RMVPFCST** command lets you specify the long SQL Name.

While it is possible on the iSeries to have a source name that is different from the object it creates, and to have a source member create many different objects, it would be very difficult for TURNOVER® for iSeries v100 to manage these items for you.

Tip: Consider the code for the **CatItemFmt** table in **Figure 1**, which shows how you can control not only the SQL and system names, but also the record format name. (This might be helpful in situations where you have RPG or COBOL programs that use native I/O.)

```
Create Table CatItemFmt
( catalogID int not null,
  productID int not null,
Primary Key( catalogID, productID ) );
Rename Table CatItemFmt to Catalog_Items for System Name CatItems;
```

Figure 1: Controlling the system AND record format names in your DDL source

About the RUNSQLSTM command's parameter requirements

You can specify additional **RUNSQLSTM** parameters in the type code create command if you need to vary from the command defaults to meet your corporate standards.

The DFTRDBCOL parameter

The **DFTRDBCOL** parameter tells the command where to place your objects and where to look for other unqualified objects being referenced. This is quite useful because the **RUNSQLSTM** is not really a compiler, and it has no concept of a library list through which it searches for unqualified objects.

The COMMIT parameter

Another parameter to consider is the **Commitment Control** (**COMMIT**) parameter. The creation of a complex SQL table, for example, may require several statements in the source. The first statement might CREATE the table and subsequent statements might apply multiple **CONSTRAINTS**. If the CREATE succeeds but one of the **CONSTRAINTS** fails, the **COMMIT** value of ***ALL** will "roll back" the prior successful statement (that is, delete the object that was created).

UNICOM Systems, Inc. Recommends

The **COMMIT** parameter is not activated in the default settings shipped with TURNOVER® for iSeries v100. If you choose to activate it, we recommend that you test it thoroughly before implementing it in a live application.

For more information about executing SQL statements using TURNOVER® for iSeries v100, see ***Managing Externally Stored Procedures*** on page 19. Also see ***Managing Programs that require*** SQL Packages

If your company uses SQL packages, then you will encounter a situation where program objects that you are promoting need to have SQL packages created or re-created.

There are two ways to handle this situation: post-run commands, or an exit program.

Post-run command method

When a *PGM on your form needs to have an SQL package created or re-created during the form run, you can link a post-run command to the form line for your program that executes the CRTSQLPKG command appropriately. Once you have done this with your program, TURNOVER® for iSeries v100 will thereafter prompt for the correct post-run command any time it encounters that object on a subsequent form.

Exit program method

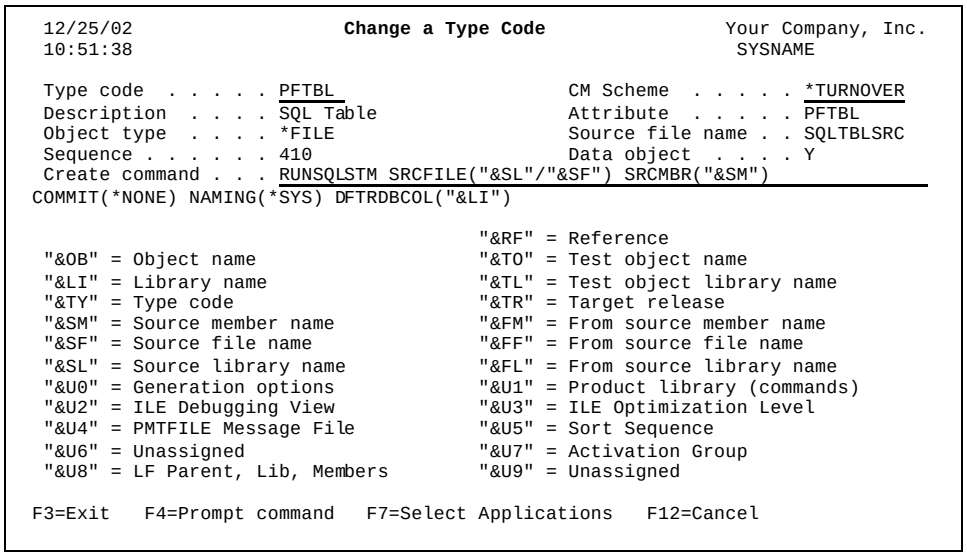
You can write an exit program to use with one of TURNOVER® for iSeries v100's form processing exit points (for example, Exit 20). Your exit program could sift through your form, identify lines eligible for CRTSQLPKG processing, and issue the command for each eligible line. This technique is best employed in situations where more automation is desired.

Executing Ad-Hoc SQL Statements on page 29.

MANAGING TABLES

An SQL table is implemented on the iSeries as a physical file. The most significant difference between a "standard" physical file and a table is the source code used to create it. To create a table, you type a "CREATE TABLE" data definition statement into a source member.

In TURNOVER® for iSeries v100, several type codes for managing SQL are shipped already defined for you. The PFTBL type code looks like this:



```
 12/25/02                    Change a Type Code                  Your Company, Inc.
 10:51:38                                                         SYSNAME

 Type code  . . . . . PFTBL                      CM Scheme  . . . . . *TURNOVER
 Description  . . . . SQL Table                  Attribute  . . . . . PFTBL
 Object type  . . . . *FILE                      Source file name . . SQLTBLSRC
 Sequence . . . . . . 410                        Data object  . . . . Y
 Create command . . . RUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM")
COMMIT(*NONE) NAMING(*SYS) DFTRDBCOL("&LI")

                                        "&RF" = Reference
 "&OB" = Object name                    "&TO" = Test object name
 "&LI" = Library name                   "&TL" = Test object library name
 "&TY" = Type code                      "&TR" = Target release
 "&SM" = Source member name             "&FM" = From source member name
 "&SF" = Source file name               "&FF" = From source file name
 "&SL" = Source library name            "&FL" = From source library name
 "&U0" = Generation options             "&U1" = Product library (commands)
 "&U2" = ILE Debugging View             "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File           "&U5" = Sort Sequence
 "&U6" = Unassigned                     "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members        "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 2: The PFTBL type code shipped with TurnOver

- The type code name *must* begin with "PF" or "LF" so that TURNOVER® for iSeries v100 knows it is a database file (copies data back, handles logical files, and so on).
- The default source file name can be modified to suit your internal naming standards.
- The table is handled like any other physical file: Data is copied back, logical files (such as Views and Indexes) are recreated, members are put back, triggers and constraints are put back; journaling is stopped and restarted, and so on.

(For more information, see ***Managing Constraints*** and ***Managing Database Security*** on pages 12 and 13.)

MANAGING INDEXES AND VIEWS

Indexes and views are implemented on the iSeries as logical files. You manage them the same as you would a table – by typing DDL statements into a source member, and creating them using the **RUNSQLSTM** command.

TURNOVER® for iSeries v100 ships the following two type codes for managing these objects:

```
  12/25/02              TurnOver Change a Type Code            Your Company, Inc.
  11:01:41                                                      SYSNAME

  Type code  . . . . . LFIDX                      CM Scheme  . . . . . *TURNOVER
  Description  . . . . SQL Index                  Attribute  . . . . . LFIDX
  Object type  . . . . *FILE                      Source file name . . SQLIDXSRC
  Sequence . . . . . . 425                        Data object  . . . . Y
  Create command . . . RUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM")
  COMMIT(*NONE)  NAMING(*SYS) DFTRDBCOL("&LI")

                                                "&RF" = Reference
  "&OB" = Object name                           "&TO" = Test object name
  "&LI" = Library name                          "&TL" = Test object library name
  "&TY" = Type code                             "&TR" = Target release
  "&SM" = Source member name                    "&FM" = From source member name
  "&SF" = Source file name                      "&FF" = From source file name
  "&SL" = Source library name                   "&FL" = From source library name
  "&U0" = Generation options                    "&U1" = Product library (commands)
  "&U2" = ILE Debugging View                    "&U3" = ILE Optimization Level
  "&U4" = PMTFILE Message File                  "&U5" = Sort Sequence
  "&U6" = Unassigned                            "&U7" = Activation Group
  "&U8" = LF Parent, Lib, Members               "&U9" = Unassigned

 F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 3: Shipped TURNOVER® for iSeries v100 type code for an SQL index

```
  12/25/02              TurnOver Change a Type Code            Your Company, Inc.
  11:01:41                                                     SYSNAME

  Type code  . . . . . LFVIEW                     CM Scheme  . . . . . *TURNOVER
  Description  . . . . SQL View                   Attribute  . . . . . LFVIEW
  Object type  . . . . *FILE                      Source file name . . SQLVIEWSRC
  Sequence . . . . . . 425                        Data object  . . . . Y
  Create command . . . RUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") COMMIT(*NONE)
 NAMING(*SYS) DFTRDBCOL("&LI")

                                                "&RF" = Reference
  "&OB" = Object name                           "&TO" = Test object name
  "&LI" = Library name                          "&TL" = Test object library name
  "&TY" = Type code                             "&TR" = Target release
  "&SM" = Source member name                    "&FM" = From source member name
  "&SF" = Source file name                      "&FF" = From source file name
  "&SL" = Source library name                   "&FL" = From source library name
  "&U0" = Generation options                    "&U1" = Product library (commands)
  "&U2" = ILE Debugging View                    "&U3" = ILE Optimization Level
  "&U4" = PMTFILE Message File                  "&U5" = Sort Sequence
  "&U6" = Unassigned                            "&U7" = Activation Group
  "&U8" = LF Parent, Lib, Members               "&U9" = Unassigned

 F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 4: Shipped TURNOVER® for iSeries v100 type code for an SQL view

- The name for the type code(s) can be anything that begins with "LF". The same information about the command parameters and source file names that applies to tables also applies to indexes and views.

- TURNOVER® for iSeries v100 manages indexes and views as it would any logical file. When a table is changed in TURNOVER® for iSeries v100, any indexes or views that are built over the table are automatically recreated.

UNICOM Systems, Inc. Recommends

You could use a single type code to manage indexes and views, called something like "LFSQL." However, we suggest using two type codes for these reasons:

1. Clarity – the type code clearly distinguishes the type of object.

2. Flexibility – using two type codes allows the source and/or objects for each to be stored in different locations.

MANAGING TRIGGERS

SQL triggers can be promoted and maintained by TURNOVER® for iSeries v100 like any other SQL object. However, triggers should also be stored in the source for the table to which they belong, so that when the table is recreated the triggers are reapplied. TURNOVER® for iSeries v100 automatically reapplies triggers that are defined with **ADDPFTRG** (such as if you were setting a trigger on a PF), but it does not reapply SQL triggers, because that would require having access to the source. Putting them in the source with the table solves that problem.

The following example shows source for both the table and the trigger:

```
Create Table Tablename
( FIELD1     Dec(7,0) Not Null,
   FIELD2     Dec(7,2) Not Null,
   FIELD3     Char(25) Not Null,
    Primary Key(FIELD1),
  Constraint FIELD2CHK
     Check(FIELD2  >= 0));

Create Trigger Triggername
          AFTER INSERT on Tablename
       (Body of Trigger statement);
```

Figure 5: Sample source for a table with a trigger

Keep in mind that it is possible to create the table without creating the trigger if there are errors in the source for the trigger. Check the joblog and the file description for details.

MANAGING CONSTRAINTS

An advantage of using SQL is its ability to specify and enforce database rules and integrity. In the source code example at the end of this document, for example, the **CUSTOMER** table has a constraint that ensures that the value entered into the **CrdLimit** column is always >= 0.

SQL also enables referential integrity through use of the "foreign key" clause in SQL. In the **SALE** table below, we have specified that a foreign key constraint exists between the **CustId** column in **SALE** and **CustId** in **CUSTOMER**. If the **CUSTOMER** row is deleted, this ensures that all matching rows in the **SALE** table are also deleted.

TURNOVER® for iSeries v100 manages both these types of constraints. You can add constraints to a table using either DDL (as illustrated on page 35) or the iSeries command interface.

UNICOM Systems, Inc. Recommends:

1. If you're using DDL source members to create your tables, include in the source for your table the DDL statements necessary for adding the desired constraints. This makes it easy for you to rebuild your database or port it to other platforms.
2. If you choose not to manage your constraints in the same DDL source as your table, then you can manipulate constraints by running the **ADDPFCST** and **RMVPFCST** commands as post-run commands on the TURNOVER® for iSeries v100 form. Because TURNOVER® for iSeries v100 handles existing constraints automatically, you should **NOT** link the commands to the creation of the corresponding object. They only need to be executed once.
3. To change or delete an existing constraint, you must use a post-run command. This also applies regardless of whether the table is built using SQL or DDS. We do recommend that you make the corresponding changes in your DDL, so that your source can always be used to recreate or port your database.
4. Add new constraints to the DDL by checking out the source, editing, and promoting back up the line.
5. We strongly recommend that you maintain a full set of all database objects at each level of your application.

For example, when you change only the **SALE** file, TURNOVER® for iSeries v100 examines the object being replaced and re-establishes the constraints with **CUSTOMER** during the form run. Again, this is another reason for having a full set of all tables, views and indexes in each environment.

Don't forget that the form is also following your data conversion instructions (such as ***MAP *DROP**). If you have the PFTBL in QA, for example, and it has some test data in it, the form will also be testing your data conversion methods. In other words, you are doing a dress rehearsal of the data conversion you will eventually be performing against live data in production. Yet another reason for maintaining objects at all levels.

MANAGING DATABASE SECURITY

DDL has statements for defining the security of your database objects. In fact, the only way that it's currently possible to define column-level authority on an iSeries database object is by using DDL statements. By default, TURNOVER® for iSeries v100 manages your security on objects using reference objects; but, because column-level security can only be applied via DDL, TURNOVER® for iSeries v100 cannot use a reference object to manage this for you.

Therefore, if you need to apply column-level security, you must assume responsibility for all the object's authority requirements. The only way to do this is to include the necessary DDL statements in your source member *and* to tell TURNOVER® for iSeries v100 NOT to manage the security by setting the ***Authority parameter*** in the type code to an 'S' (use the system defaults). Because you're now managing the security for the object, you must include the DDL statements necessary to define all levels of appropriate authority for the object – including its object-level authorities.

UNICOM Systems, Inc. Recommends

1. We recommend against using column-level security and managing the security settings in the source. IBM documentation suggests that there is a significant performance hit associated with column-level security. You can often use views with object authority to get the same effect as column-level authority.

2. If you must use column-level security, one method of automating this in TURNOVER® for iSeries v100 would be to create a special type code (for example, PFTBLS). It would be an exact duplicate of the PFTBL type code, EXCEPT that you would override the type code in the application definition at all levels to force the ***Authority parameter*** to 'S'.

MANAGING FUNCTIONS AND SQL STORED PROCEDURES

Important!

1. Managing functions and SQL stored procedures requires that you use a TURNOVER® for iSeries v100 feature called a *CM scheme*. You must be running TURNOVER® Release 5.3 to use CM schemes. (Directions follow.)

2. Type codes for these items are NOT shipped with TURNOVER® for iSeries v100; you must create them as instructed below. You can use either the **RUNSQLSTM** command, OR a special TURNOVER® for iSeries v100 version called **TRUNSQLSTM**, for the create command for these type codes. Which one you choose depends on the requirements you have for the creation of the object in the target library. Read ***Choosing a create command for functions and SQL stored procedures*** on page 16.

Using SQL, you can create both executable functions and stored procedures. You can write these procedures using either a high-level language such as RPGLE, or by using SQL's Procedural Language (SPL).

When you use the **RUNSQLSTM** command against this source, SQL first translates the source into ILE C with embedded SQL. It is then precompiled using the **CRTSQLCI** command to generate the C code with the embedded SQL. Next, it uses the **CRTCMOD** command to create the ILE module. Finally, it uses the **CRTPGM** (for procedures) or the **CRTSRVPGM** (for functions) to bind the module into a program.

At the same time, an entry is created in the SQL Catalog. This is a group of views over a set of system dictionary files stored in library QSYS2. These views contain additional information about SQL tables, views, indexes, keys, constraints, user-defined types, functions and procedures.

If you attempt to create a function or an SQL stored procedure that is already defined in this table, however, you would receive an error stating that that routine already exists. This means that before every run, you must execute the **DROP PROCEDURE** statement. Of course, if the function or SQL stored procedure that you are dropping does not exist, you'll receive another error.

TURNOVER® for iSeries v100 takes all of this hassle away from you! One of the enhancements made in TURNOVER® Release 5.3 was the ability for you to create your own Change Management Scheme. For more information on this feature, see the TURNOVER® for iSeries v100 Supplement entitled ***Managing User-Defined Object Types*** (**#59**).

In a nutshell, a CM scheme lets you precisely control what commands are used for iSeries functions such as **CHKOBJ**, **CMPSRC**, **DLTOBJ**, **GRTOBJAUT**, **MOVOBJ**, and so on. TURNOVER® for iSeries v100 references these commands as it performs its work.

We have created a special scheme just for managing SQL functions and SQL stored procedures that are written using SPL. This scheme checks for the existence of the SQL stored procedure in the catalog and updates it accordingly. This effectively automates the process, lifting it from the shoulders of your I.T. staff.

The scheme invokes several new programs that help you manage these objects. The SQLDLTOBJ and SQLMOVOBJ programs run an SQL statement to drop a function or an SQL stored procedure after the program object is moved or deleted. SQLMOVOBJ also has special logic to handle form recovery, such that the SQL definition is retrieved and stored in the archive library before the existing object is moved out. If the form goes into recovery, the saved definition (rather than moving the object) is used to recreate the function or SQL stored procedure.

Your TURNOVER® for iSeries v100 installation may or may not already have this scheme defined. To see what CM schemes are already defined, run the **TWRKCMSCH** command from a TURNOVER® for iSeries v100 command line (there are no parameters). If you do not see the SQLCM scheme, you must restore it using the instructions provided in ***Restoring the SQLCM scheme*** below. If the scheme exists, you should at least verify that its configuration matches the requirements outlined in this document.

Restoring the SQLCM scheme

To restore the SQLCM scheme, perform the following steps:

1. Search for a save file named SQLSCH in one of the TURNOVER® for iSeries v100 libraries (it is shipped in SOFTTURN by default). If you cannot find it, contact Technical Support to receive this save file. The programs that support the methods of the scheme are already loaded in SOFTTURN and do not need to be restored.

2. From a TURNOVER® for iSeries v100 command line, run the following command to restore the scheme from the save file:

   ```
   TRSTCMSCH SAVF(SOFTTURN/SQLSCH) TOSCHEME(*SAVSCHEME)
   REPLACE(*NO) RSTPGMS(*NO) PGMLIB(*SCHEME) RSTRSRCLIB(*NO)
   RSRCLIB(*SAVLIB)
   ```

3. If you have loaded TURNOVER® for iSeries v100 into a library other than SOFTTURN, you must change where the scheme will look for the programs that run any methods specifying the default program and library names:

   ```
   TCHGCMSCH SCHEME(SQL) DESC('SQL Scheme')
   DFTPGM(your_lib_name/*TURNOVER)
   ```

4. From a TURNOVER® for iSeries v100 command line, run the **TWRKCMSCH** command to verify your results.

Creating the type codes

Figure 6 through **Figure 9** show how your SQL function and SQL stored procedure type codes should look. In particular, note the following settings:

- You can name the type code anything you like.
- The ***CM scheme*** points TURNOVER® for iSeries v100 to the proper scheme.
- The ***Object type*** for an SQL stored procedure must be ***PGM**.
- The ***Object Type*** for a function must be ***SRVPGM**.
- A "**Y**" value for ***Data object*** allows the application definition to "explode" data, which effectively copies the object to multiple libraries during a single promotion.
- A value of 800 for ***Sequence*** ensures that the form has already processed other SQL items, such as tables and views, before the function or SQL procedure is created.
- The ***Create command*** can be either the **RUNSQLSTM** command or the **TRUNSQLSTM** create command. (The next topic tells you how to choose one of these commands.)

Choosing a create command for functions and SQL stored procedures

There are times when you need to use the **NAMING(*SYS)** and **DFTRDBCOL(*NONE)** parameter values in your **RUNSQLSTM** create command, so that the SQL runtime will use the library list to resolve object references.

However, when you don't specify a default database collection (that is, when **DFTRDBCOL** is set to ***NONE**), TURNOVER® for iSeries v100 loses control of where SQL builds the objects that your script creates.

To resolve this issue, TURNOVER® for iSeries v100 uses its own version of the **RUNSQLSTM** command. It is called **TRUNSQLSTM**, and it has the same parameters as **RUNSQLSTM**, along with a few additions (specifically the added **CURLIB** parameter, which you need in this case. Another parameter called **RUNAS** is also important for other reasons; read about that in ***Controlling unauthorized database changes*** on page 32.)

When you specify the "&LI" substitution variable as the value for the **CURLIB** parameter, and combine this with the **NAMING(*SYS)** and **DFTRDBCOL(*NONE)** parameter values in your type code create command, TURNOVER® for iSeries v100 resolves the substitution variable from the application definition before submitting the command, allowing SQL to create the objects in a location where object references will resolve successfully.

Examples of type code definitions that use both **RUNSQLSTM** and **TRUNSQLSTM** follow.

```
 12/25/02                    Change a Type Code                  Your Company, Inc.
 11:01:41                                                        SYSNAME

 Type code  . . . . . SQLFNC                    CM Scheme  . . . . . SQL
 Description  . . . . SQL Function              Attribute  . . . . . SQL
 Object type  . . . . *SRVPGM                   Source file name . . QSQLSRC
 Sequence . . . . . . 800                       Data object  . . . . Y
 Create command . . . RUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") COMMIT(*NONE)
 NAMING(*SQL) DFTRDBCOL("&LI")

                                          "&RF" = Reference
 "&OB" = Object name                      "&TO" = Test object name
 "&LI" = Library name                     "&TL" = Test object library name
 "&TY" = Type code                        "&TR" = Target release
 "&SM" = Source member name               "&FM" = From source member name
 "&SF" = Source file name                 "&FF" = From source file name
 "&SL" = Source library name              "&FL" = From source library name
 "&U0" = Generation options               "&U1" = Product library (commands)
 "&U2" = ILE Debugging View               "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File             "&U5" = Sort Sequence
 "&U6" = Unassigned                       "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members          "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 6: Example of a type code for creating an SQL function with RUNSQLSTM

```
 12/25/02                    Change a Type Code                  Your Company, Inc.
 11:01:41                                                        SYSNAME

 Type code  . . . . . SQLFNC                    CM Scheme  . . . . . SQL
 Description  . . . . SQL Function              Attribute  . . . . . SQL
 Object type  . . . . *SRVPGM                   Source file name . . QSQLSRC
 Sequence . . . . . . 800                       Data object  . . . . Y
Create command . . . SOFTTURNE/TRUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") CO
MMIT(*NONE) NAMING(*SYS) DFTRDBCOL(*NONE) CURLIB("&LI")

                                          "&RF" = Reference
 "&OB" = Object name                      "&TO" = Test object name
 "&LI" = Library name                     "&TL" = Test object library name
 "&TY" = Type code                        "&TR" = Target release
 "&SM" = Source member name               "&FM" = From source member name
 "&SF" = Source file name                 "&FF" = From source file name
 "&SL" = Source library name              "&FL" = From source library name
 "&U0" = Generation options               "&U1" = Product library (commands)
 "&U2" = ILE Debugging View               "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File             "&U5" = Sort Sequence
 "&U6" = Unassigned                       "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members          "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 7: Type code for creating an SQL function with TRUNSQLSTM

```
 12/25/02                    Change a Type Code                  Your Company, Inc.
 11:01:41                                                        SYSNAME

 Type code  . . . . . SQLPRC                    CM Scheme  . . . . . SQL
 Description  . . . . SQL Stored Procedure      Attribute  . . . . . SQL
 Object type  . . . . *PGM                      Source file name . . QSQLSRC
 Sequence . . . . . . 800                       Data object  . . . . Y
 Create command . . . RUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") COMMIT(*NONE)
 NAMING(*SQL) DFTRDBCOL("&LI")
                                    "&RF" = Reference
 "&OB" = Object name                "&TO" = Test object name
 "&LI" = Library name               "&TL" = Test object library name
 "&TY" = Type code                  "&TR" = Target release
 "&SM" = Source member name         "&FM" = From source member name
 "&SF" = Source file name           "&FF" = From source file name
 "&SL" = Source library name        "&FL" = From source library name
 "&U0" = Generation options         "&U1" = Product library (commands)
 "&U2" = ILE Debugging View         "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File       "&U5" = Sort Sequence
 "&U6" = Unassigned                 "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members    "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 8: Type code for creating an SQL stored procedure with RUNSQLSTM

```
 12/25/02                    Change a Type Code                  Your Company, Inc.
 11:01:41                                                        SYSNAME

 Type code  . . . . . SQLPRC                    CM Scheme  . . . . . SQL
 Description  . . . . SQL Stored Procedure      Attribute  . . . . . SQL
 Object type  . . . . *PGM                      Source file name . . QSQLSRC
 Sequence . . . . . . 800                       Data object  . . . . Y
Create command . . . SOFTTURNE/TRUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") CO
MMIT(*NONE) NAMING(*SYS) DFTRDBCOL(*NONE) CURLIB("&LI")
                                    "&RF" = Reference
 "&OB" = Object name                "&TO" = Test object name
 "&LI" = Library name               "&TL" = Test object library name
 "&TY" = Type code                  "&TR" = Target release
 "&SM" = Source member name         "&FM" = From source member name
 "&SF" = Source file name           "&FF" = From source file name
 "&SL" = Source library name        "&FL" = From source library name
 "&U0" = Generation options         "&U1" = Product library (commands)
 "&U2" = ILE Debugging View         "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File       "&U5" = Sort Sequence
 "&U6" = Unassigned                 "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members    "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 9: Type code for creating an SQL stored procedure with TRUNSQLSTM

MANAGING EXTERNALLY STORED PROCEDURES AND FUNCTIONS

Important!

- Managing externally stored procedures and functions requires that you use a TURNOVER® for iSeries v100 feature called a *CM scheme*. You must be running TURNOVER® Release 5.3 to use CM schemes. (Directions follow.)

- Type codes for these items are NOT shipped with TURNOVER® for iSeries v100; you must create them as instructed below. You can use either the **RUNSQLSTM** command, OR a special TURNOVER® for iSeries v100 version called **TRUNSQLSTM**, for the create command for these type codes. Which one you choose depends on the requirements you have for the creation of the object in the target library. Read ***Choosing a create command for externally stored procedures*** on page 23.

You write an externally stored procedure or function in one of the programming languages on the iSeries. You can compile the host language programs to create ***PGM** or ***SRVPGM** objects. To create an externally stored procedure or function, you must compile the source code for the host language to create a program object.

For TURNOVER® for iSeries v100 to properly promote your routine's catalog entries, you ***must*** do the following:

1. Create a source member that uses the **CREATE PROCEDURE** or **CREATE FUNCTION** statements, to be run with the **RUNSQLSTM** or **TRUNSQLSTM** commands.

2. Give your source member the same name as the program that it references.

3. Use the **SPECIFIC** statement to give the routine's catalog entries the same name as the source member name.

Type codes for external procedures and functions

TURNOVER® for iSeries v100 manages externally stored procedures and functions using multiple type codes, as follows:

- One type code to manage the ***PGM** or ***SRVPGM** object.
- Another type code to manage creation of the catalog entries from a source member.

It is possible to change the logic of an externally stored procedure or function without having to change the catalog entry, as long as the parameters and object attributes don't change. Therefore, you may not have to promote the catalog entries each time you modify the program. However, if you make a change that requires the catalog entry to be re-created, be sure to check out and modify the **CREATE PROCEDURE** or **CREATE FUNCTION** source as well.

You can use any appropriate type code (such as RPGLE or SQLRPI) to manage the ***PGM** object and the current processes that TURNOVER® for iSeries v100 uses will still be the same. To manage the catalog entries, you must use a TURNOVER® for iSeries v100 change management scheme. For information about managing catalog entries, see ***Managing catalog entries with a TURNOVER® for iSeries*** v100 CM scheme on page 21.

Managing catalog entries with a TURNOVER® for iSeries v100 CM scheme

Using a TURNOVER® for iSeries v100 Change Management Scheme, you can precisely control what commands are used for iSeries functions such as **CHKOBJ**, **DLTOBJ**, **MOVOBJ**, and so on. TURNOVER® for iSeries v100 references these commands as it performs its work.

We have created a special scheme just for managing the catalog entries of externally stored procedures and functions. This scheme overrides certain methods to handle the catalog entries:

- The ***CHKOBJ** method examines the catalog entries for a specified library to let you know if a catalog entry exists for the externally stored procedure or function.
- The ***DLTOBJ** method issues the **DROP PROCEDURE** or **DROP FUNCTION** statement to remove the catalog entry for an externally stored procedure or function.
- The ***MOVOBJ** method is used during archiving of existing catalog entries, during distribution, and during recovery processing, as follows:
    - For archiving and distribution, this method ensures that you have source for the existing catalog entries. It does so by generating SQL DDL source from the existing catalog entries in the archive or distribution library. This provides a safeguard against promoting catalog entries that may have been created without source through iSeries navigator.
    - For archiving, this method also drops the existing catalog entries to allow you to create new entries.
    - For recovery processing, this method ensures that the catalog entries are recreated properly from archived source, or from source that may have been automatically generated for archive.
- The ***RTVOBJD** method returns certain information about a catalog entry, such as the date and time that it was created, the description of the entry (from the **COMMENT** statement), and so forth.
- Methods ***CHGOBJ**, ***CHGOBJOWN**, ***CRTDUPOBJ**, and ***GRTOBJAUT** are set to ***NONE** to perform no processing, the reason being that these methods perform tasks that are not relevant to catalog entries.

Your TURNOVER® for iSeries v100 installation may or may not already have this scheme defined. To see what CM schemes are already defined, run the **TWRKCMSCH** command from a TURNOVER® for iSeries v100 command line (there are no parameters). If you do not see the EXTPRC scheme, you must restore it using the instructions provided in ***Restoring the EXTPRC scheme***, below. If the scheme exists, you should at least verify that its configuration matches the requirements outlined in this document.

Important!

If your tape date precedes November 2004, then you must apply cumulative changes before you can use the support for external functions.

Restoring the EXTPRC scheme

To restore the EXTPRC scheme, perform the following steps:

1. Search for a save file named UTEXTPRC in one of the TURNOVER® for iSeries v100 libraries (it is shipped in SOFTTURN by default). If you cannot find it, contact Technical Support to receive this save file. The programs that support the methods of the scheme are already loaded in SOFTTURN and do not need to be restored.[1]

2. From a TURNOVER® for iSeries v100 command line, run the following command to restore the scheme from the save file:

   ```
   TRSTCMSCH SAVF(SOFTTURN/UTEXTPRC) TOSCHEME(*SAVSCHEME)
   REPLACE(*NO) RSTPGMS(*NO) PGMLIB(*SCHEME) RSTRSRCLIB(*NO)
   RSRCLIB(*SAVLIB)
   ```

3. If you have loaded TURNOVER® for iSeries v100 into a library other than SOFTTURN, you must change where the scheme will look for the programs that run any methods specifying the default program and library names:

   ```
   TCHGCMSCH SCHEME(EXTPRC) DESC('External Stored Procedure
   Scheme') DFTPGM(your_lib_name/*TURNOVER)
   ```

4. From a TURNOVER® for iSeries v100 command line, run the **TWRKCMSCH** command to verify your results.

[1] Programs are included in TURNOVER® Release 5.3, with a tape date of June 2004.

Creating the type codes

Figure 10, Figure 11 and **Figure 11** show how your externally stored procedures and functions type code should look. In particular, note the following settings:

- You should name the type code for external procedures EXTPRC, and the type code for external functions EXTFUN, but you can name the type code anything you like. If you plan to use an alternate type code, see ***TURNOVER® for iSeries v100 cross reference*** on page 26.
- The ***CM scheme*** points TURNOVER® for iSeries v100 to the proper scheme.
- The ***Object type*** for an externally stored procedure must be ***EXTPRC** and externally stored functions must be *EXTFUN.
- A "**Y**" value for ***Data object*** allows the application definition to "explode" data, which creates the catalog entries for multiple libraries during a single promotion (optional).
- A value of 800 for ***Sequence*** ensures that the form has already processed the externally stored procedure or function program, before the catalog entries are created.
- The ***Create command*** can be either the **RUNSQLSTM** command or the **TRUNSQLSTM** create command. (The next topic tells you how to choose one of these commands.)

Choosing a create command for externally stored procedures and functions

There are times when you need to use the **NAMING(*SYS)** and **DFTRDBCOL(*NONE)** parameter values in your **RUNSQLSTM** create command, so that the SQL runtime will use the library list to resolve object references.

However, when you don't specify a default database collection (that is, when **DFTRDBCOL** is set to ***NONE**), TURNOVER® for iSeries v100 loses control of where SQL builds the objects that your script creates.

To resolve this issue, TURNOVER® for iSeries v100 uses its own version of the **RUNSQLSTM** command. It is called **TRUNSQLSTM**, and it has the same parameters as **RUNSQLSTM**, along with a few additions (specifically the added **CURLIB** parameter, which you need in this case. Another parameter called **RUNAS** is also important for other reasons; read about that in ***Controlling unauthorized database changes*** on page 32.)

When you specify the "&LI" substitution variable as the value for the **CURLIB** parameter, and combine this with the **NAMING(*SYS)** and **DFTRDBCOL(*NONE)** parameter values in your type code create command, TURNOVER® for iSeries v100 resolves the substitution variable from the application definition before submitting the command, allowing SQL to create the objects in a location where object references will resolve successfully.

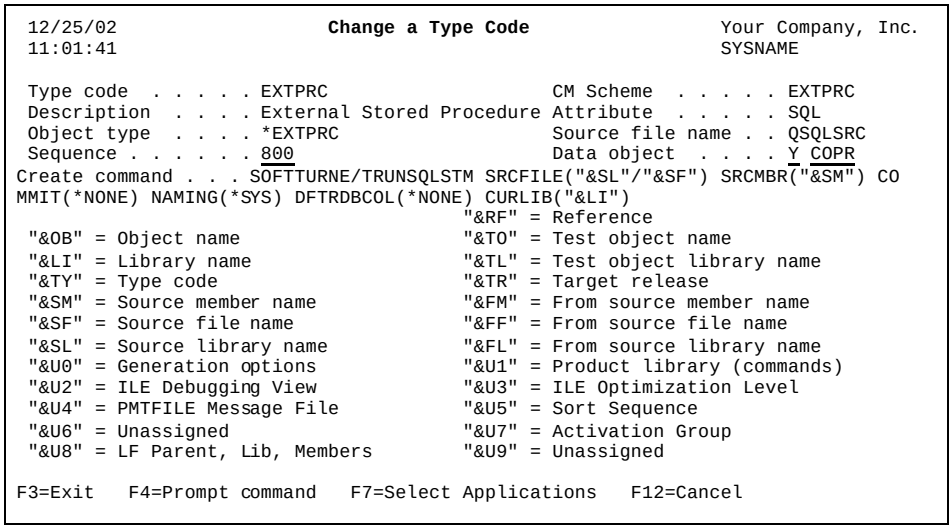
Examples of type code definitions that use both **RUNSQLSTM** and **TRUNSQLSTM** follow (below and on the next page):

```
 12/25/02                    Change a Type Code                  Your Company, Inc.
 11:01:41                                                        SYSNAME

 Type code  . . . . . EXTPRC                     CM Scheme  . . . . . EXTPRC
 Description  . . . . External Stored Procedure  Attribute  . . . . . SQL
 Object type  . . . . *EXTPRC                    Source file name . . QSQLSRC
 Sequence . . . . . 800                          Data object  . . . . Y COPR
 Create command . . . RUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") COMMIT(*NONE)
 NAMING(*SQL) DFTRDBCOL("&LI")
                                              "&RF" = Reference
 "&OB" = Object name                          "&TO" = Test object name
 "&LI" = Library name                         "&TL" = Test object library name
 "&TY" = Type code                            "&TR" = Target release
 "&SM" = Source member name                   "&FM" = From source member name
 "&SF" = Source file name                     "&FF" = From source file name
 "&SL" = Source library name                  "&FL" = From source library name
 "&U0" = Generation options                   "&U1" = Product library (commands)
 "&U2" = ILE Debugging View                   "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File                 "&U5" = Sort Sequence
 "&U6" = Unassigned                           "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members              "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 10: Type code for creating an externally stored procedure with RUNSQLSTM

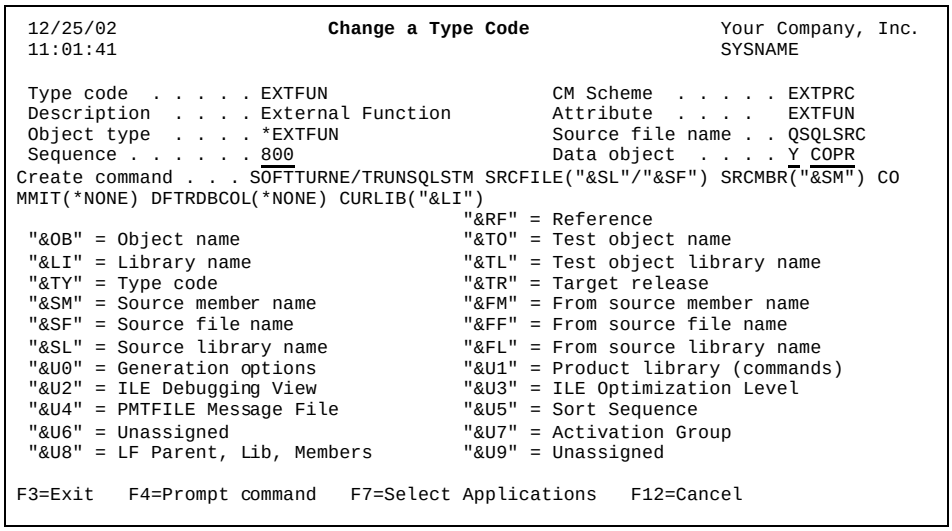


```
 12/25/02                  Change a Type Code                 Your Company, Inc.
 11:01:41                                                     SYSNAME

 Type code  . . . . . EXTPRC                   CM Scheme  . . . . . EXTPRC
 Description  . . . . External Stored Procedure Attribute  . . . . . SQL
 Object type  . . . . *EXTPRC                  Source file name . . QSQLSRC
 Sequence . . . . . 800                        Data object  . . . . Y COPR
Create command . . . SOFTTURNE/TRUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") CO
MMIT(*NONE) NAMING(*SYS) DFTRDBCOL(*NONE) CURLIB("&LI")
                                          "&RF" = Reference
 "&OB" = Object name                      "&TO" = Test object name
 "&LI" = Library name                     "&TL" = Test object library name
 "&TY" = Type code                        "&TR" = Target release
 "&SM" = Source member name               "&FM" = From source member name
 "&SF" = Source file name                 "&FF" = From source file name
 "&SL" = Source library name              "&FL" = From source library name
 "&U0" = Generation options               "&U1" = Product library (commands)
 "&U2" = ILE Debugging View               "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File             "&U5" = Sort Sequence
 "&U6" = Unassigned                       "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members          "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 11: Type code for creating an externally stored procedure with TRUNSQLSTM



```
 12/25/02                  Change a Type Code                 Your Company, Inc.
 11:01:41                                                     SYSNAME

 Type code  . . . . . EXTFUN                   CM Scheme  . . . . . EXTPRC
 Description  . . . . External Function        Attribute  . . . .   EXTFUN
 Object type  . . . . *EXTFUN                  Source file name . . QSQLSRC
 Sequence . . . . . 800                        Data object  . . . . Y COPR
Create command . . . SOFTTURNE/TRUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") CO
MMIT(*NONE) DFTRDBCOL(*NONE) CURLIB("&LI")
                                          "&RF" = Reference
 "&OB" = Object name                      "&TO" = Test object name
 "&LI" = Library name                     "&TL" = Test object library name
 "&TY" = Type code                        "&TR" = Target release
 "&SM" = Source member name               "&FM" = From source member name
 "&SF" = Source file name                 "&FF" = From source file name
 "&SL" = Source library name              "&FL" = From source library name
 "&U0" = Generation options               "&U1" = Product library (commands)
 "&U2" = ILE Debugging View               "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File             "&U5" = Sort Sequence
 "&U6" = Unassigned                       "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members          "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 12: Type code for creating an external function with TRUNSQLSTM

Service program libraries

When using a service program (*SRVPGM) as the external procedure or function, the service program library name must be hard coded. (This is an IBM limitation.) This makes promotion of procedures and functions that use these a bit more difficult as normal TURNOVER® for iSeries v100 processing will not modify your source code. To handle this situation, the **TRUNSQLSTM** command has been enhanced to have the additional parameter Replace Library in Source (RPLLIB).

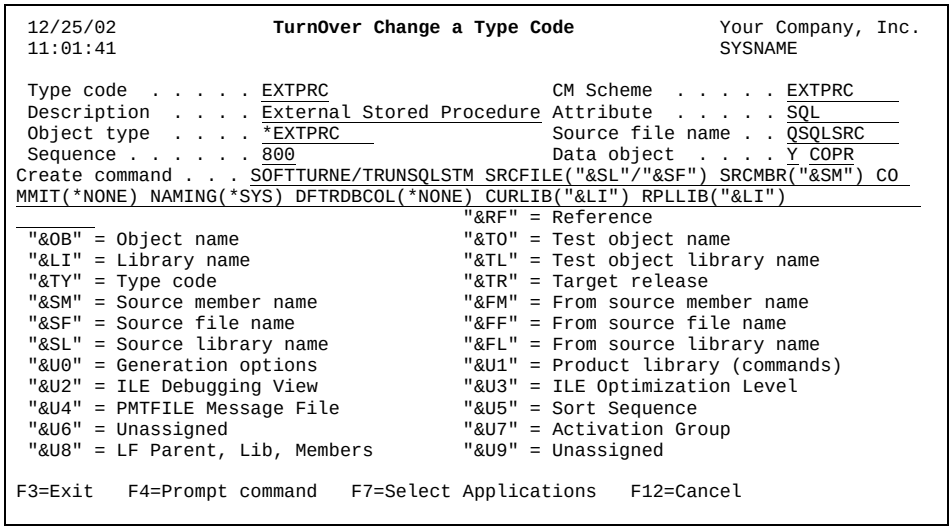
Using the RPLLIB parameter on the **TRUNSQLSTM** command and the place holder &LIBRARY in your source code causes the TRUNSQLSTM process to copy your source code into a source file in QTEMP, and then the source in QTEMP is modified to replace the &LIBRARY with the actual library name.

For example, your source code may contain this:

```
…
EXTERNAL NAME '&LIBRARY/OURSRVPGM(PROCEDURE01)'
…
```

When the **TRUNSQLSTM** command is run, the source is copied to QTEMP and any occurrence of '&LIBRARY' is replaced with the actual library name before the statements are executed.

You would then define the RPLLIB parameter in your global type code definition, using the appropriate substitution variable as shown in the next screen print:



```
 12/25/02              TurnOver Change a Type Code              Your Company, Inc.
 11:01:41                                                       SYSNAME

 Type code  . . . . . EXTPRC                    CM Scheme  . . . . . EXTPRC
 Description  . . . . External Stored Procedure Attribute  . . . . . SQL
 Object type  . . . . *EXTPRC                   Source file name . . QSQLSRC
 Sequence . . . . . . 800                       Data object  . . . . Y COPR
Create command . . . SOFTTURNE/TRUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") CO
MMIT(*NONE) NAMING(*SYS) DFTRDBCOL(*NONE) CURLIB("&LI") RPLLIB("&LI")
                                                "&RF" = Reference
 "&OB" = Object name                            "&TO" = Test object name
 "&LI" = Library name                           "&TL" = Test object library name
 "&TY" = Type code                              "&TR" = Target release
 "&SM" = Source member name                     "&FM" = From source member name
 "&SF" = Source file name                       "&FF" = From source file name
 "&SL" = Source library name                    "&FL" = From source library name
 "&U0" = Generation options                     "&U1" = Product library (commands)
 "&U2" = ILE Debugging View                     "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File                   "&U5" = Sort Sequence
 "&U6" = Unassigned                             "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members                "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

TURNOVER® for iSeries v100 cross reference

When you use TURNOVER® for iSeries v100 to promote a program that has been defined as an externally stored procedure or function, as shown in this document, TURNOVER® for iSeries v100 adds records to the TURNOVER® for iSeries v100 cross-reference table showing the program as being used by the catalog entry. **WRKOBJREF**, or option **15** from the worklist, will show the catalog entry as a PGM-CALL from the program. Although we recommend the type code EXTPRC with an object type of *EXTPRC for procedures and type code EXTFUN with an object type of *EXTFUN for functions, you can use a different type code, as long as the global type code is defined with the object type *EXTPRC or *EXTFUN.

Things to remember

Things that you should keep in mind while managing externally stored procedures or functions are the following:

- Create the global type code as defined in this document. Assign the type code to all the necessary applications and configure it to use method **CSCO** (**Copy Source**, **Compile Object**).

- If you distribute, make sure the applications are setup to distribute source for the type codes EXTPRC and EXTFUN. Also, ensure that the remote machines have the type codes defined globally with the EXTPRC scheme, and that the type codes have been assigned to the correct levels of the remote application. The remote application should also use method **CSCO** for promotion.

- Any type code that you create to manage procedures or functions must have the global object type for those type codes set to *EXTPRC or *EXTFUN.

- Use a source member to define the **CREATE PROCEDURE** or **CREATE FUNCTION** statements, and make use of the **SPECIFIC** statement. The source member name and the **SPECIFIC** name should match. Do not use library names as qualifiers in this source – the **TRUNSQLSTM** command will set up the ***CURLIB** correctly. If you do need to use library names as qualifiers (as is the case with *SRVPGMs), make use of the RPLLIB parameter on the TRUNSQLSTM command and use &LIBRARY instead of the actual library name.

- Promotion will archive existing source. If the existing catalog entry does not have source, it will be generated in the archive library into a file called QARCHIVSQL.

- TURNOVER® for iSeries v100 cross-reference table entries for the ***EXTPRC** and ***EXTFUN** objects come from the ***PGM** object. Promotion of just the ***EXTPRC** or ***EXTFUN** object will not update the cross-reference table entries. You must promote (or run the cross-reference) for the ***PGM** to see the references. TURNOVER® for iSeries v100 does not provide cross-reference table entries for ***EXTPRC** or ***EXTFUN** objects that use *SRVPGMs.

MANAGING MATERIALIZED QUERY TABLES

Materialized query table (MQTs) provide another method of improving performance of your queries. An MQT is a table that contains the results of a previously run query, along with the query's definition. It provides a mechanism for improving the response time of complex SQL queries. While it can be thought of as a summary table, what sets an MQT apart from a regular summary table is the fact that the SQE optimizer is aware of it and its relationship to the query and base tables that were specified when it was created and populated. This means that the optimizer considers using the MQT in the access plan of subsequent similar queries if it determines that it is appropriate to do so. Because the MQT is already created, populated, joined, aggregated, and sorted, this can result in significant performance improvements for complex queries.

It is important to understand that an MQT is a table (a physical file with an object type of *FILE and an attribute of PF) that resides in a library (schema) in the System i environment.

While their potential performance efficiencies are appealing for DBAs, query developers, and report users, MQTs have several attributes and limitations that must be understood prior to implementation. Probably most important, you must know that MQTs are not automatically maintained. This means that as the base tables used to populate the MQT change, the data in the MQT does not also change.

The functionality of an MQT is similar to the role of an index. Both objects provide a path to the data that the user is normally unaware of. Unlike an index, a user might directly query the MQT just like a table or view.

Additional information concerning Materialized Query Tables can be found in the IBM Redbook manual titled; *"Getting Started with DB2 Web Query for System i* - SG247214" and the IBM white paper; *"Creating and using materialized query tables (MQT) in IBM DB2 for i5/OS Version 2"*.

Based on the above information our recommendations for handling **Materialized Query Tables** within TURNOVER® for iSeries v100 is to setup a global type code named **LFMQT** (similar to LFVIEW). In doing so, it will eliminate the adding of a COPR form line each time the form is edited or run. Prior to the September 2009 release of TURNOVER® for iSeries v100, using the “LF" prefixed in the type code created a Warning during the form run processing when TURNOVER® for iSeries v100 attempted to change the logical file because it was assuming it was a physical file. This condition was corrected with that, and subsequent releases of TURNOVER®.

MANAGING PROGRAMS THAT REQUIRE SQL PACKAGES

If your company uses SQL packages, then you will encounter a situation where program objects that you are promoting need to have SQL packages created or re-created.

There are two ways to handle this situation: post-run commands, or an exit program.

Post-run command method

When a *PGM on your form needs to have an SQL package created or re-created during the form run, you can link a post-run command to the form line for your program that executes the CRTSQLPKG command appropriately. Once you have done this with your program, TURNOVER® for iSeries v100 will thereafter prompt for the correct post-run command any time it encounters that object on a subsequent form.

Exit program method

You can write an exit program to use with one of TURNOVER® for iSeries v100's form processing exit points (for example, Exit 20). Your exit program could sift through your form, identify lines eligible for CRTSQLPKG processing, and issue the command for each eligible line. This technique is best employed in situations where more automation is desired.

EXECUTING AD-HOC SQL STATEMENTS

There will be times when you need to execute an SQL statement to make a change, but there is no "object movement" associated with this change. For example, you might want to use the **ALTER TABLE** statement to add a new constraint, or change the column headings of a table.

To execute these ad hoc SQL statements, use the SQLSTM type code that is shipped with TURNOVER® (as of Release 5.2):

```
 12/25/02              TurnOver Change a Type Code              Your Company, Inc.
 11:01:41                                                       SYSNAME

 Type code  . . . . . SQLSTM                    CM Scheme  . . . . . *TURNOVER
 Description  . . . . SQL Statements            Attribute  . . . . . SQL
 Object type  . . . . *SQLSTM                   Source file name . . QSQLSRC
 Sequence . . . . . . 800                       Data object  . . . . Y
 Create command . . . RUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM") COMMIT(*NONE)
 NAMING(*SQL) DFTRDBCOL("&LI")

                                          "&RF" = Reference
 "&OB" = Object name                      "&TO" = Test object name
 "&LI" = Library name                     "&TL" = Test object library name
 "&TY" = Type code                        "&TR" = Target release
 "&SM" = Source member name               "&FM" = From source member name
 "&SF" = Source file name                 "&FF" = From source file name
 "&SL" = Source library name              "&FL" = From source library name
 "&U0" = Generation options               "&U1" = Product library (commands)
 "&U2" = ILE Debugging View               "&U3" = ILE Optimization Level
 "&U4" = PMTFILE Message File             "&U5" = Sort Sequence
 "&U6" = Unassigned                       "&U7" = Activation Group
 "&U8" = LF Parent, Lib, Members          "&U9" = Unassigned

F3=Exit   F4=Prompt command   F7=Select Applications   F12=Cancel
```

Figure 13: Example of the type code for Ad-Hoc SQL statements

Remember that any source member containing such an ad-hoc SQL statement will by default be executed as written, at *every* application level and on *every* machine where this form will run. If you do not want this statement to run multiple times on the same machine, or if you only want this statement to run on certain machines, you must manually handle those scenarios by editing the form before it runs to remove that line, or by not running the form or not distributing it to the remote.

Now do the following:

- Use **F6** on the worklist to add a new item. Fill in the new item's name and description; the type code will be SQLSTM.
- Check the item out to reserve its name and to allow promotion.
- In your development environment, type the SQL source code into the member.
- **Be careful about using option 36**; it will execute the SQL source. (Remember that for SQL objects, this is not a compile; it is actually executing the SQL source using the **RUNSQLSTM** command.)
- Build and run your form.

Notes:

1. If your application distributes and you want this SQL source code to execute on your remotes as well, you must define this type code to distribute the source member and use a promotion method of CSCO on the remote computer. This also means, of course, that the SQL licensed programs must be installed on the remote machines.

2. The sequence number of "800" means that this statement is one of the last items that is processed by this form. This helps ensure that other items that might be referenced by this ad-hoc SQL statement, such as tables, are already in place.

3. Realize that this promotion might be performing any number of a variety of activities by executing the SQL statements in the source file. TURNOVER® for iSeries v100 has no way of knowing what these statements are doing, so there is no way to "undo" the effects of processing the statement(s) TURNOVER® for iSeries v100 executes using this technique.

Controlling unauthorized database changes

A TURNOVER® for iSeries v100 command called **TRUNSQLSTM** allows you to control the user profile under which people can run ad hoc SQL statements. This command contains the basic important parameters of the **RUNSQLSTM** command, but adds some additional parameters and logic to it, to help ensure database integrity. To use this command, specify it in a TURNOVER® for iSeries v100 type code, like this:

```
10/06/03                    Change a Type Code                  YOUR COMPANY INC.
14:07:26                                                        YOURSYS

Type code  . . . . . SQLSTT                      CM scheme  . . . . . *TURNOVER
Description  . . . . SQLSTM- TRUNSQLSTM          Attribute  . . . . . SQL
Object type  . . . . *SQLSTM                     Source file name . . QSQLSRC
Sequence . . . . . . 800                         Data object  . . . . Y *DEFAULT
Create command . . . SOFTTURNE/TRUNSQLSTM SRCFILE("&SL"/"&SF") SRCMBR("&SM")
COMMIT(*NONE) DFTRDBCOL("&LI") RUNAS("&U9")

                                                "&RF" = Reference     "&X0"-"&X9" = Exit
"&OB" = Object name                             "&TO" = Test object name
"&LI" = Library name                            "&TL" = Test object library name
"&TY" = Type code                               "&TR" = Target release
"&SM" = Source member name                      "&FM" = From source member name
"&SF" = Source file name                        "&FF" = From source file name
"&SL" = Source library name                     "&FL" = From source library name
"&U0" = Generation options                      "&U1" = Debugging views
"&U2" = Optimization level                      "&U3" = Compiler options
"&U4" = Activation Group                        "&U5" = WEBLICATE System designator
"&U6" = WEBLICATE Release                       "&U7" = Service program source file
"&U8" = Binding directory                       "&U9" = Module source file
F3=Exit  F4=Prompt/select   F7=Applications  F12=Cancel  F22=Long command
```

This command MUST be qualified when used in the type code definition because it resides in the language library, which typically is not in the library list during a TURNOVER® for iSeries v100 form run.

This command builds and executes the **RUNSQLSTM** command from within a program that does not use adopted authority. In this way, you can override the adopted authority present during a TURNOVER® for iSeries v100 form. We've supplied three additional parameters:

Run as user profile (RUNAS)

Indicates the user profile under whose authority settings you want the **RUNSQLSTM** command to run.

Use a type code substitution variable for this parameter at the different levels of your application. At lower levels of the application, use a value of ***CURRENT**; at higher levels, specify a value such as QSECOFR or the application owner.

UNICOM Systems, Inc. Recommends

Add an entry for the **RUNAS** keyword into the SOFTTURND/TKEYPRTF file. This file is where you record command keyword entries that should never be overridden. The entry in the TKEYPRTF file should look like this:

```
KKTYPE:  *ALL
KKKEY:   RUNAS
```

This entry ensures that **RUNAS(*CURRENT)** cannot be overridden at the lower levels of your application.

Extra Default Parms (EXTRADFT)

When defining the **TRUNSQLSTM** command on a global type code, use this field to include any additional **RUNSQLSTM** command parameter values that should be treated as global defaults.

Extra User-Specified Parms (EXTRAUSER)

When a programmer prompts the **TRUNSQLSTM** command in TURNOVER® for iSeries v100 at compile time, s/he should use this field to supply any additional non-global, **RUNSQLSTM** command parameter values that should be used for the compile.

DISTRIBUTING SQL CHANGES

When distributing SQL, you must "compile" on all systems to execute the SQL script.

These instructions assume that you are just distributing objects to the remote and using **MO** (**Move Object**) as your promotion method. If you are already distributing source code and compiling on the remote, you might already be setup correctly.

Follow these instructions:

1. Ensure that every SQL type code in the distributing level of the sending application is set to distribute source. Most applications that distribute have the ***Distribute Source*** parameter set to "N," so you must edit each SQL type code in the distributing application level and make sure this setting is "Y."

2. On the remote (production) computer, have TURNOVER® for iSeries v100 store the source that it receives into library QTEMP. To do this, change the ***Source Target Library*** value to QTEMP in the corresponding level of the remote application definition that is used to receive and run the remote form. This trick automatically deletes the source code from the Production machine after the form completes and it is no longer needed for the creation of the object.

3. If you prefer to leave a copy of the scripts on the remote system, specify a source file and library instead of QTEMP.

4. Also, you must override every SQL type code in the remote application definition to use method **CSCO**. This forces the recreation of the objects from the source. For this to work, the IBM SQL licensed programs must be installed on the production machine as well. This recommendation ensures that TURNOVER® for iSeries v100 accurately creates the objects on the target computer, maintaining all constraints and triggers exactly as they were defined and named on the development machine.

5. On the remote computer, verify that your application library list is correct for compiling.

A WORD ABOUT OBJECT CREATION SEQUENCE

Your work in development (for example, from the worklist) is largely up to you. In other words, you decide what should compile first (the parent or base table), what second (the child or related table, views and indexes). If you were creating the objects in the examples below for the first time in development, you would need to compile the objects in the following order:

1. Compile CUSTOMER and PART first because they are the parent files.

2. Compile SALE next because of the foreign key constraint on CUSTOMER.

3. Compile CUSTERR next because it's the logical view over CUSTOMER.

4. Compile SALEITEM next because it has foreign keys over both CUSTOMER and SALE.

5. Compile CUSSALPRT last because it is a logical view over CUSTOMER, SALE, and PART.

When you promote these items to QA, TURNOVER® for iSeries v100 looks at the compile dates and times of the objects and orders them on the form according to that (within type code, of course). This makes the form "smart" enough to create the items in the proper order.

EXAMPLE: DDL SOURCE FOR A SIMPLE DATABASE

This is an example of some DDL source members. Each source member is separated by "======". The name of the source member must be the same as the name of the object it will create.

```
=========================================
Create Table Customer
( CustID     Dec(    7, 0 ) Not Null,
  Name       Char(  30    ) Not Null,
  ShpLine1   Char( 100    ) Not Null,
  ShpLine2   Char( 100    ) Not Null,
  ShpCity    Char(  30    ) Not Null,
  ShpState   Char(   2    ) Not Null,
  ShpPsCd1   Char(  10    ) Not Null,
  ShpPsCd2   Char(  10    ) Not Null,
  ShpCntry   Char(  30    ) Not Null,
  PhnVoice   Char(  15    ) Not Null,
  PhnFax     Char(  15    ) Not Null,
  PhnExt     Char(   4    ) Not Null,
  Territory  Char(   1    ) Not Null,
  Status     Char(   1    ) Not Null
                            With Default ' ',
  CrdLimit   Dec(    7, 0 ) With Default
Null,
  EntDate    Date           Not Null,
Primary Key( CustID ),
Constraint StatusChk
   Check(Status IN (' ', '1', '2', '3')),
Constraint TerritoryChk
   Check(Territory IN ('1', '2', '3')),
Constraint CreditChk
   Check( CrdLimit >= 0 ) )
=========================================

Create Table Part
( PartID     Dec(    7, 0 ) Not Null,
  PartDesc   Char(  50    ) Not Null,
Primary Key( PartID ) )
=========================================
Create Table Sale
( OrderID    Dec(    7, 0 ) Not Null,
  SaleDate   Date           Not Null,
  SaleTot    Dec(    7, 2 ) Not Null,
  CustID     Dec(    7, 0 ) Not Null,
  TotPrice   Dec(    7, 2 ) Not Null,
Primary Key( OrderID ),
Constraint SalesCustFK Foreign Key( CustID )
   References Customer( CustID )
   On Delete Cascade  )
```

```
=========================================
Create Table SaleItem
( OrderID    Dec(    7, 0 ) Not Null,
  PartID     Dec(    7, 0 ) Not Null,
  Quantity   Dec(    7, 0 ) Not Null,
  Price      Dec(    7, 2 ) Not Null,
  Discount   Dec(    7, 2 ) Not Null,
Primary Key( OrderID, PartID ),
Constraint DiscountChk
   Check( Discount <= Price ),
Constraint SlItmOrdFK Foreign Key( OrderID )
   References Sale( OrderID )
   On Delete Cascade
   On Update Restrict,
Constraint SlItmPrtFK Foreign Key( PartID )
   References Part( PartID )
   On Delete Restrict
   On Update Restrict );

Grant Select,
      Update( Quantity,
              Price,
              Discount)
      On SaleItem
      To GrpAcctng;

Grant Select,
      Update( Quantity,
              Discount)
      On SaleItem
      To GrpSales;
=========================================

Create Index CustTerr
 On Customer
 ( Territory,
   Name )
=========================================

Create View CusSalPrt
(  CustID,
   Name,
   SaleDate,
   PartDesc )
As Select Customer.CustID,
          Name,
          SaleDate,
          PartDesc
   From   Customer,
          Sale,
          SaleItem,
          Part
   Where  Customer.CustID = Sale.CustID
     And  Sale.OrderID = SaleItem.OrderID
     And  SaleItem.PartID = Part.PartID
```

SUMMARY

Having read this information, you should be able to configure TURNOVER® for iSeries v100 to manage iSeries database objects using either DDS source or SQL source. You should be able to configure TURNOVER® for iSeries v100 to build database components with appropriate SQL object names, and execute the **RUNSQLSTM** command with the necessary parameter values. You should be able to run a TURNOVER® for iSeries v100 promotion that correctly builds database indexes, tables, and views. You should know how to structure your TURNOVER® for iSeries v100 application to ensure that appropriate database constraints and security settings are properly applied to your SQL objects in production. Additionally, you should be able to execute ad hoc SQL statements using special TURNOVER® for iSeries v100 type codes, and manage stored functions and procedures. Finally, you should be able to construct and test your TURNOVER® for iSeries v100 setup for SQL objects before implementing it in your live environment.

In other words, you should now be able to apply all the power that is TURNOVER® for iSeries v100 to better manage your SQL objects.

If you have any questions about the information in this document, please contact a UNICOM Systems, Inc. Technical Support Representative by phone, fax, or email at the locations shown at the beginning of this document.