IBM

Cognitive Systems

Breaking out of PASE: Accessing the rest of your IBM i from Python

Kevin Adler - kadler@us.ibm.com - @kadler_ibm

Session ID: 170414

Agenda Key: 41CM

Two Main Methods

- Database access
- XMLSERVICE

ibm_db DB2 interface

- ibmdb project: https://github.com/ibmdb/python-ibmdb
- Cross-platform, open source project sponsored by IBM.
- Supports DB2 LUW, Informix, and DB2 for i using DB2 Connect on Windows, Mac OS X, Linux, and a few other platforms
- Ported to run in PASE, using PASE CLI by IBM Rochester
- Uses the Apache 2 License
-
  ```
  pip3 install /QOpenSys/QIBM/ProdData/OPS/Python-pkgs/ibm_db/ibm_db-*cp34*.whl
  ```

Not Just a Simple Database Interface

- ibm_db – main database interface with two sub-modules

  ibm_db
  - DB2-specific, “raw” database interface
  - written in C, on top of DB2 Connect CLI or IBM i CLI
  - https://github.com/ibmdb/python-ibmdb/wiki/APIs

  ibm_db_dbi
  - Python Database API Specification 2.0 compliant interface
  - written in Python, on top of ibm_db
  - https://www.python.org/dev/peps/pep-0249
- ibm_db_django – database interface for Django web framework. Django is a web framework based on the Model-View-Controller pattern and is comparable to Ruby on Rails

Not Just a Simple Database Interface

- ibm_db_sa* – SQLAlchemy adapter for ibm_db. SQLAlchemy is an object relational mapper (ORM), similar to Hibernate for Java or Ruby on Rails' ActiveRecord
- ibm_db_alembic* – Alembic interface for ibm_db. Alembic is a database migration tool for SQLAlchemy which allows keeping database changes consistent with application changes, similar to ActiveRecord::Migration in Ruby on Rails

- `pip3 install /QOpenSys/QIBM/ProdData/OPS/Python-pkgs/ibm_db/ibm_db-*cp34*.whl`
- `pip3 install /QOpenSys/QIBM/ProdData/OPS/Python-pkgs/ibm_db/ibm_db_django-*cp34*.whl`

* not currently supported

PEP 249

- Standard Python DB API: https://www.python.org/dev/peps/pep-0249/
- Mostly documented there, though connect() is DB specific

ibm_db_dbi Global Functions

- apilevel
  - string constant indicating supported Python DB API level
  - returns “2.0”
- threadsafety
  - integer constant indicating the thread safety supported
  - returns 0 indicating no thread safety
- paramstyle
  - string constant indicating the type of parameter marker format
  - returns “qmark” to indicate question marks are used for parameter markers eg. *select * from sysibm.sysdummy1 where a=? and b=?*
- **connect and pconnect functions**
  - main entry point to the module
  - parameters are database-dependent
  - return a Connection object

Connecting

- connect(dsn, user, password, host, database, conn_options)
  - dsn – data source name string (`"Database=*LOCAL;UID=kadler"`)
    - alternative to separate parameters
    - supports DATABASE, UID, and PWD keywords, equating to database, user, and password parameters
    - defaults to empty string
  - user and password – authentication info, defaults to empty string
  - host – not supported (used on DB2 Connect / LUW)
  - database – RDB to connect to eg. *LOCAL, defaults to empty string
  - conn_options – dictionary of connection options (ibm_db_dbi.set_option) to set during connection, defaults to None
- pconnect
  - persistent connection: not closed when close() is called
  - stored in a hash for retrieval during future pconnect()
- Using defaults for all parameters results in connecting to the local database as the current user
  - `conn = ibm_db_dbi.connect()`

Connecting

```
import ibm_db_dbi as db2

# Connect to *LOCAL as current user
conn = db2.connect()

# Connect to REMOTE as current user
conn = db2.connect(database='REMOTE')

# Connect to *LOCAL as a anon
conn = db2.connect(user='anon', \
                   password='secr3t')
```

Connection Object Methods

- close()
  - close db connection immediately
  - automatically called when object is deleted
  - pconnect objects goes back to pool instead
- commit()
  - commit a pending transaction
  - autocommit is defaulted to off
- rollback()
  - rollback a pending transaction
  - closing a connection without committing causes an implicit rollback
- **cursor()**
  - returns a new Cursor object

Closing Connections

```
import ibm_db_dbi as db2
conn = db2.connect()
conn.close() # rollback automatically performed
if True:
    conn2 = db2.connect()
# conn2 automatically closed and rolled back
```

Connection Objects Methods (ibm_db_dbi extensions)

- dbms_name attribute
  - gets the SQL_DBMS_NAME name (“AS” on IBM i)
- dbms_ver attribute
  - gets the SQL_DBMS_VER (eg. “07020” for IBM i 7.2)
- set_autocommit(is_on)
  - enable or disable autocommit
- get_option(attr)
  - returns the value for the given attribute
  - (calls SQLGetConnectAttr)
- set_option(attributes)
  - attributes is a dictionary of attributes to set
  - (calls SQLSetConnectAttr)

Connection Object Methods (ibm_db_dbi metadata extensions)

- tables(schema, table)
  - retrieves the tables existing in the given table
  - schema and table default to None
  - *currently doesn't support non-uppercase schema and table names
- indexes(only_unique, schema, table)
  - retrieves index information for the given table
  - only_unique can be set to True or False to only return unique indexes
  - schema and table default to None, only_unique defaults to True
  - *currently doesn't support non-uppercase schema and table names
- columns(schema, table, column)
  - retrieves column information for the given table and column names
  - schema, table, and column default to None
  - *currently doesn't support non-uppercase schema and table names
  - *converts all matching column names to lower case, if column not None

Connection Object Methods (ibm_db_dbi metadata extensions)

- primary_keys(*unique, schema, table)
  - retrieves primary key information for the given table
  - schema, and table default to None
  - *unique is not used or needed
- foreign_keys(*unique, schema, table)
  - retrieves foreign key information for the given table
  - unique, schema, and table default to None
  - *unique is not used or needed

Cursor Objects

- description – read-only attribute
  - “2d” sequence describing the columns
  - each index contains a 7-item sequence for that column in this order:
    - name – column name
    - type_code – ibm_db_dbi type object
    - display_size – how many characters needed to display
    - internal_size – how many bytes needed to store
    - precision – total digits for NUMERIC/DECIMAL columns
    - scale – digits after decimal for NUMERIC/DECIMAL/TIMESTAMP columns
    - null_ok – is the column nullable?

Type Classes

- Date(year, month, day)
- Time(hour, minute, second)
- Timestamp(year, month, day, hour, minute, second)
- Binary(string)
- TimeFromTicks(ticks)
- TimestampFromTicks(ticks)

Cursor objects

```
import ibm_db_dbi as db2
conn = db2.connect()
c1 = conn.cursor()
c1.execute("create table t(c char(10) not null, d decimal(10,5), x XML)")
c1.execute("select * from t")
d = c1.description
print(d[0][6]) # False (nullable)
print(d[0][0]) # 'C' (column name)
print(d[1][4]) # 10 (precision)
print(d[2][1]) # db2.XML (column type)
```

Type Objects

ibm_db_dbi Type Object	DB2 data type
STRING	CHAR, VARCHAR, *BINARY, ...*
TEXT	CLOB, DBCLOB
XML	XML
BINARY	BLOB
NUMBER	INTEGER, SMALLINT
BIGINT	BIGINT
FLOAT	FLOAT, REAL, DOUBLE, DECFLOAT
DECIMAL	DECIMAL, NUMERIC
DATE	DATE
TIME	TIME
DATETIME	TIMESTAMP
ROWID	(not currently mapped)

Cursor objects

- rowcount – read only attribute
  - # of rows inserted, updated, or deleted by last execute call
  - NOT the number of rows in a cursor/result set (select)
- callproc(name, param_tuple)
  - call the given procedure with the given parameters
  - returns modified arguments (INOUT/OUT parameters)
- execute(query, param_tuple)
  - execute the query with the given parameters
- executemany(query, tuple_of_param_tuple)
  - need to pass a tuple of tuples containing the arguments
  - same as execute(), but does “block insert”
  - emulated on IBM i by multiple calls to SQLExecute

Cursor objects

```
import ibm_db_dbi as db2
conn = db2.connect()
c1 = conn.cursor()
c1.execute("select * from qiws.qcustcdt")
print(c1.rowcount) # -1

o = c1.callproc("myproc", ("in", "out", "inout"))
print(o) # ('in', "5", 'tuoni')

c1.execute("update t set d=1.1 where c <> c")
print(c1.rowcount) # 0
```

Cursor objects

- arraysize – read/write attribute
  - sets the cursor array size
- fetchone()
  - fetch the next row from the cursor or None if there are no more
- fetchmany(size)
  - fetch the next set of rows from the cursor or None if there are no more
  - fetches up to *size* rows
  - if no size is specified, returns *arraysize* rows instead
- fetchall()
  - fetch all the remaining rows from the cursor or None if there are no more
- fetchone() returns a list of column values, while fetchman() and fetchall() return a list of rows containing a list of column values

Cursor objects

```
import ibm_db_dbi as db2
conn = db2.connect()
c1 = conn.cursor()
c1.execute("select * from qiws.qcustcdt")
row = c1.fetchone()
print(row[2] + ' ' + row[1]) # G K Henning
rows = c1.fetchmany(2)
print(len(rows)) # 2
print(rows[0][2] + ' ' + rows[0][1]) # B D Jones
print(rows[1][2] + ' ' + rows[1][1]) # S S Vine
rows = c1.fetchall()
print(len(rows)) # 9
```

Cursor objects

- nextset()
  - position to the next result set returned by a stored procedure
  - returns None if there are no more result sets
- setinputsizes(size_list)
  - predefine memory areas for input parameters
  - currently does nothing on ibm_db_dbi
- setoutputsize(size, col)
  - specify the max size of an output parameter
  - currently does nothing on ibm_db_dbi
- next() and __iter__()
  - allows iterating over a cursor using the standard Python syntax for other sequence types

Cursor objects

```
import ibm_db_dbi as db2
conn = db2.connect()
c1 = conn.cursor()
c1.execute("select * from qiws.qcustcdt")
for row in c1:
    print(row)
```

Cursor objects

```
import ibm_db_dbi as db2
conn = db2.connect()
c1 = conn.cursor()
c1.callproc("proc_returns_3_rs")

rs = True
while rs:
    for row in c1:
        print(row)

    rs = c1.nextset()
```

Example: Converting database table to text table

```
from prettytable import from_db_cursor
import ibm_db_dbi as db2

conn = db2.connect()

cur = conn.cursor()
cur.execute("select * from qiws.qcustcdt")
print(from_db_cursor(cur))
```

Example: Converting database table to text table

```
+--------+----------+------+---------------+--------+-------+--------+--------+--------+---------+--------+
| CUSNUM |  LSTNAM  | INIT |     STREET    |  CITY  | STATE | ZIPCOD | CDTLMT | CHGCOD |  BALDUE | CDTDUE |
+--------+----------+------+---------------+--------+-------+--------+--------+--------+---------+--------+
| 938472 | Henning  | G K  | 4859 Elm Ave  | Dallas |   TX  | 75217  |  5000  |   3    |  37.00  |  0.00  |
| 839283 | Jones    | B D  | 21B NW 135 St | Clay   |   NY  | 13041  |  400   |   1    |  100.00 |  0.00  |
| 392859 | Vine     | S S  | PO Box 79     | Broton |   VT  |  5046  |  700   |   1    |  439.00 |  0.00  |
| 938485 | Johnson  | J A  | 3 Alpine Way  | Helen  |   GA  | 30545  |  9999  |   2    | 3987.50 | 33.50  |
| 397267 | Tyron    | W E  | 13 Myrtle Dr  | Hector |   NY  | 14841  |  1000  |   1    |   0.00  |  0.00  |
| 389572 | Stevens  | K L  | 208 Snow Pass | Denver |   CO  | 80226  |  400   |   1    |  58.75  |  1.50  |
| 846283 | Alison   | J S  | 787 Lake Dr   | Isle   |   MN  | 56342  |  5000  |   3    |  10.00  |  0.00  |
| 475938 | Doe      | J W  | 59 Archer Rd  | Sutter |   CA  | 95685  |  700   |   2    |  250.00 | 100.00 |
| 693829 | Thomas   | A N  | 3 Dove Circle | Casper |   WY  | 82609  |  9999  |   2    |   0.00  |  0.00  |
| 593029 | Williams | E D  | 485 SE 2 Ave  | Dallas |   TX  | 75218  |  200   |   1    |  25.00  |  0.00  |
| 192837 | Lee      | F L  | 5963 Oak St   | Hector |   NY  | 14841  |  700   |   2    |  489.50 |  0.50  |
| 583990 | Abraham  | M T  | 392 Mill St   | Isle   |   MN  | 56342  |  9999  |   3    |  500.00 |  0.00  |
+--------+----------+------+---------------+--------+-------+--------+--------+--------+---------+--------+
```

Example: Converting database table to Excel spreadsheet

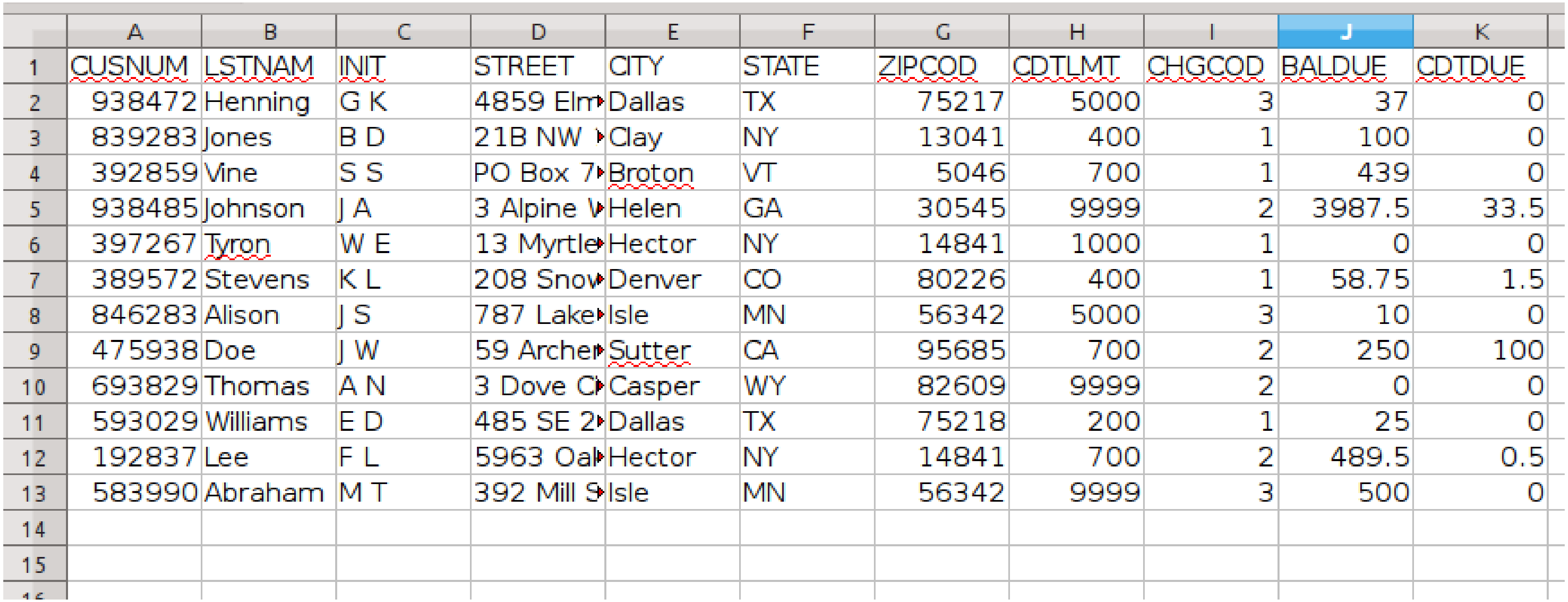
```
from xlsxwriter import Workbook
import ibm_db_dbi as db2

conn = db2.connect()

cur = conn.cursor()
cur.execute("select * from qiws.qcustcdt")
headers = [descr[0] for descr in cur.description]

with Workbook('qcustcdt.xlsx') as workbook:
    worksheet = workbook.add_worksheet()
    worksheet.write_row('A1', headers)
    for rownum, row in enumerate(cur, start=1):
        worksheet.write_row(rownum, 0, row)
```

Example: Converting database table to Excel spreadsheet



	A	B	C	D	E	F	G	H	I	J	K
1	CUSNUM	LSTNAM	INIT	STREET	CITY	STATE	ZIPCOD	CDTLMT	CHGCOD	BALDUE	CDTDUE
2	938472	Henning	G K	4859 Elm	Dallas	TX	75217	5000	3	37	0
3	839283	Jones	B D	21B NW	Clay	NY	13041	400	1	100	0
4	392859	Vine	S S	PO Box 7	Broton	VT	5046	700	1	439	0
5	938485	Johnson	J A	3 Alpine W	Helen	GA	30545	9999	2	3987.5	33.5
6	397267	Tyron	W E	13 Myrtle	Hector	NY	14841	1000	1	0	0
7	389572	Stevens	K L	208 Snow	Denver	CO	80226	400	1	58.75	1.5
8	846283	Alison	J S	787 Lake	Isle	MN	56342	5000	3	10	0
9	475938	Doe	J W	59 Archer	Sutter	CA	95685	700	2	250	100
10	693829	Thomas	A N	3 Dove Ci	Casper	WY	82609	9999	2	0	0
11	593029	Williams	E D	485 SE 2	Dallas	TX	75218	200	1	25	0
12	192837	Lee	F L	5963 Oak	Hector	NY	14841	700	2	489.5	0.5
13	583990	Abraham	M T	392 Mill S	Isle	MN	56342	9999	3	500	0
14											
15											

Python + XMLSERVICE = itoolkit

itoolkit – an XMLSERVICE wrapper

- itoolkit project: https://ibm.biz/itoolkitpython
- Python interface to XMLService: https://ibm.biz/xmlservice
- Let's you call
  - RPG programs and service programs
  - CL commands
  - PASE programs and shell scripts
  - SQL database access
- `pip3 install /QOpenSys/QIBM/ProdData/OPS/Python-pkgs/itoolkit/itoolkit-*cp34*.whl`
- Verify it's installed by running

```
python3 -c "import itoolkit;
print(itoolkit.__version__)"
```

What Can We Do?

- Commands
  - iCmd – Call CL command (even with output parameters)
  - iCmd5250 – call CL command and get screen output
- Programs
  - iPgm – Call program
  - iSrvPgm – Call service program
- Database
  - iSqlQuery – call DB2 query
  - iSqlPrepare, iSqlExecute
  - iSqlParm
  - iSqlFetch
- iSh – call PASE program or shell script
- iXml – anything else

Which Transport is Right for You?

- iLibCall
  - no configuration needed
  - fastest call, directly from your job, but no isolation from bad calls
  - local connection only
  - some things don't work from chroot
- iDB2Call
  - calls over ibm_db
  - no configuration needed
  - bad calls kill QSQSRVR job, not your job
  - local and remote connections supported
- iRestCall
  - need to configure XMLSERVICE in one of your HTTP servers
  - bad calls kill your FastCGI job, not your job
  - local and remote connections supported
  - need ~~SSL~~ TLS for security

Output Parameters from Commands

```
# In Apache config:
ScriptAlias /cgi-bin/ /QSYS.LIB/QXMLSERV.LIB/
<Directory /QSYS.LIB/QXMLSERV.LIB/>
AllowOverride None
 order allow,deny
 allow from all
 SetHandler cgi-script
 Options +ExecCGI
</Directory>
# QXMLSERV (IBM) can be replaced by XMLSERVICE
(download), ZENDSVR6 (php), POWERRUBY, etc.
```

Choosing the Right XMLSERVICE

Choose which XMLSERVICE library you want for iDB2Call, iLibCall set the XMLSERVICE environment variable before calling Python:

```
export XMLSERVICE=QXMLSERV
# or ...
export XMLSERVICE=XMLSERVICE
# or ...
export XMLSERVICE=ZENDSVR6
# ... and so on
# or from within Python:
import os
os.environ["XMLSERVICE"] = "QXMLSERV" # ...
```

Connecting

```
# iLibCall example
from itoolkit import *
from itoolkit.lib.ilibcall import *

itransport = iLibCall()

# iRestCall example
from itoolkit import *
from itoolkit.rest.irestcall import *

itransport = iRestCall(url, user, password)
```

Connecting

```
# iDB2Call example
from itoolkit import *
from itoolkit.db2.idb2call import *


itransport = iDB2Call(user, password)
# or
conn = ibm_db.connect(database, user, password)
itransport = iDB2Call(conn)
```

Basics

```
from itoolkit import *
from itoolkit.lib.ilibcall import *
itransport = iLibCall()
itool = iToolKit()
itool.add( ... ) # op1
itool.add( ... ) # op2
itool.call(itransport)
op1 = itool.dict_out('op1')
if 'success' in op1:
    print (op1['success'])
else:
    print (op1['error'])
```

iPgm and iSrvPgm Details

```
iPgm(key, name, options)
iSrvPgm(key, name, function, options)
```

- key: unique key to identify output, arbitrary
- name: program or service program name
- function: function to call in the service program
- options: dictionary of options (optional)
  - error ('on', 'off', or 'fast'): 'on' causes script to stop on error, 'off' gives joblog info, 'fast' gives brief error log
  - lib: IBM i library name, defaults to *LIBL
  - function: function to call in service program
  - mode ('opm' or 'ile')

```
iPgm('foo', 'FOO', {'function': 'bar' }) ==
iSrvPgm('foo', 'FOO', 'bar')
```

Example: Calling an ILE program from Python

```
from itoolkit import *
from itoolkit.lib.ilibcall import *
itransport = iLibCall()
itool = iToolKit()
itool.add(iCmd('addlible', 'addlible KADLER'))
itool.add(
  iPgm('my_key','MYPGM')
    .addParm(iData('inchara','1a','a'))
    .addParm(iDS('INDS1')
        .addData(iData('dscharb','1a','b'))
        .addData(iData('desdec','12p2','3.33'))
  )
)
itool.call(itransport)
mypgm_results = itool.dict_out('my_key')
```

iCmd and iCmd5250 Details

```
iCmd(key, cmd_string, options)
iCmd5250(key, cmd_string, options)
```

- key: unique key to identify output, arbitrary
- cmd_string: CL command string
- options: dictionary of options (optional)
  - error ('on', 'off', or 'fast'): 'on' causes script to stop on error, 'off' gives joblog info, 'fast' gives brief error log
  - exec ('cmd', 'system', 'rexx') – only supported for iCmd, defaults to 'rexx' for commands with output parms and 'cmd' otherwise
    - 'cmd' runs command through QCMDEXC
    - 'system' runs command through PASE system command, required to get display output
    - 'rexx' runs command through REXX, required for output parms

Output Parameters from Commands

```
itool.add(iCmd('rtvjoba', 'RTVJOBA USRLIBL(?)
SYSLIBL(?) CCSID(?N) OUTQ(?)'))
itool.call(itransport)
rtvjoba = itool.dict_out('rtvjoba')
if 'success' in rtvjoba:
    print(rtvjoba['row'][0]['USRLIBL'])
    print(rtvjoba['row'][1]['SYSLIBL'])
    print(rtvjoba['row'][2]['CCSID'])
    print(rtvjoba['row'][3]['OUTQ'])
```

Output Parameters from Commands

```
itool = iToolKit(irow=0)
itool.add(iCmd('rtvjoba', 'RTVJOBA USRLIBL(?)
SYSLIBL(?) CCSID(?N) OUTQ(?)'))
itool.call(itransport)
rtvjoba = itool.dict_out('rtvjoba')
if 'success' in rtvjoba:
    print(rtvjoba['USRLIBL'])
    print(rtvjoba['SYSLIBL'])
    print(rtvjoba['CCSID'])
    print(rtvjoba['OUTQ'])
```

https://bitbucket.org/litmis/python-itoolkit/issues/3/improve-irow-handling-for-5250-output

Command Display Output

```
itool.add(iCmd5250('wrkactjob', 'WRKACTJOB'))
itool.call(itransport)
wrkactjob = itool.dict_out('wrkactjob')
print(wrkactjob['wrkactjob'])
```

Command Display Output

```
5770SS1 V7R1M0 100423          Work with Active Jobs
Reset . . . . . . . . . . . . . . . . . :   *NO
Subsystems  . . . . . . . . . . . . . . :   *ALL
CPU Percent Limit . . . . . . . . . . . :   *NONE
Response Time Limit . . . . . . . . . . :   *NONE
Sequence  . . . . . . . . . . . . . . . :   *SBS
Job name  . . . . . . . . . . . . . . . :   *ALL
CPU %  . . . :      .0          Elapsed time . . . .
Subsystem/Job  User        Number   User        Type
QBATCH         QSYS        218043   QSYS        SBS
QCMN           QSYS        218046   QSYS        SBS
  QACSOTP      QUSER       218066   QUSER       PJ
  QLZPSERV     QUSER       218081   QUSER       PJ
```

iSh Details

```
iSh(key, cmd, options)
```

- key: unique key to identify output, arbitrary
- cmd: command string to execute
- options: dictionary of options (optional)
  - error ('on', 'off', or 'fast'): 'on' causes script to stop on error, 'off' gives joblog info, 'fast' gives brief error log
  - row: ('on', 'off'): wraps stdout in row field

iSh

```
itool.add(iSh('ciphers', "ssh -Q cipher | xargs
echo | sed 's| |,|g'"))
itool.call(itransport)
ciphers = itool.dict_out('ciphers')
print(ciphers['ciphers'])
# 3des-cbc,blowfish-cbc,cast128-
cbc,arcfour,arcfour128,arcfour256,aes128-
cbc,aes192-cbc,aes256-cbc,rijndael-
cbc@lysator.liu.se,aes128-ctr,aes192-ctr,aes256-
ctr,chacha20-poly1305@openssh.com
```

iSh

```
itool.add(iSh('ciphers', "ssh -Q cipher"))
itool.call(itransport)
ciphers = itool.dict_out('ciphers')
print(','.join(ciphers['ciphers'].splitlines()))
# 3des-cbc,blowfish-cbc,cast128-
cbc,arcfour,arcfour128,arcfour256,aes128-
cbc,aes192-cbc,aes256-cbc,rijndael-
cbc@lysator.liu.se,aes128-ctr,aes192-ctr,aes256-
ctr,chacha20-poly1305@openssh.com

# 0.08s vs 0.34s for sed version, why?
```

Calling SQL

```
query = "select job_name, elapsed_cpu_time,
elapsed_time from table(qsys2.active_job_info()) x
where subsystem = 'QHTTPSVR' fetch first 3 rows only"

itool.add(iSqlQuery('query', query))
itool.add(iSqlFetch('fetch'))
itool.add(iSqlFree('free')) #more important now
itool.call(tran)
for row in itool.dict_out('fetch')['row']:
    print(row)
time.sleep(1)
itool.call(tran)
for row in itool.dict_out('fetch')['row']:
    print(row)
```

No Elapsed Time?

```
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'776966/QSYS/QHTTPSVR'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787491/QTMHHTTP/ADMIN'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787504/QTMHHTTP/ADMIN'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787518/QTMHHTTP/ADMIN'}
```

```
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'776966/QSYS/QHTTPSVR'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787491/QTMHHTTP/ADMIN'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787504/QTMHHTTP/ADMIN'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787518/QTMHHTTP/ADMIN'}
```

What State are You In?

- XMLSERVICE state*less* by default
- Nothing preserved between calls
- New SQLConnect every time, QSQSRVR job reset
- RPG programs and activation groups go away
- What do you do if you need statefulness?

Stateful SQL

```
query = "select job_name, elapsed_cpu_time,
elapsed_time from table(qsys2.active_job_info()) x
where subsystem = 'QHTTPSVR' fetch first 3 rows only"
tran = iLibCall(ictl='*sbmjob', ipc='/uniqepath')
itool.add(iSqlQuery('query', query))
itool.add(iSqlFetch('fetch'))
itool.add(iSqlFree('free')) # more important now
itool.call(tran)
for row in itool.dict_out('fetch')['row']:
    print(row)
time.sleep(1)
itool.call(tran)
for row in itool.dict_out('fetch')['row']:
    print(row)
```

Stateful Example

```
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'776966/QSYS/QHTTPSVR'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787491/QTMHHTTP/ADMIN'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787504/QTMHHTTP/ADMIN'}
{'ELAPSED_TIME': '0.000', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787518/QTMHHTTP/ADMIN'}

{'ELAPSED_TIME': '1.281', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'776966/QSYS/QHTTPSVR'}
{'ELAPSED_TIME': '1.281', 'ELAPSED_CPU_TIME': '10', 'JOB_NAME':
'787491/QTMHHTTP/ADMIN'}
{'ELAPSED_TIME': '1.281', 'ELAPSED_CPU_TIME': '50', 'JOB_NAME':
'787504/QTMHHTTP/ADMIN'}
{'ELAPSED_TIME': '1.281', 'ELAPSED_CPU_TIME': '0', 'JOB_NAME':
'787518/QTMHHTTP/ADMIN'}
```

XMLSERVICE Control Keywords

- XMLSERVICE has CTL keywords, like H spec in RPG
- Specified using ictl parameter on iLibCall, iDB2Call, or iRestCall constructor
- http://yips.idevcloud.com/wiki/index.php/XMLService/XMLSERVICEQuick#ctl
- Defaults to '*here' if not specified

XMLSERVICE Control Keywords

- *here – run inside XMLSERVICE job
- *sbmjob – submit new job and run there
- *call – define timeout to wait for XMLSERVICE calls to complete before returning to client (MSGW workaround)
- *idle – define idle timeout to end XMLSERVICE stateful job when hasn't been used in a while
- *debug[proc,cgi] – stop job in MSGW to start debugging

Stateful Needs IPC

- To run stateful, you need to define *sbmjob ctl option
- Also need to define IPC path
  - path used to communicate with submitted job, whatever you like
  - per session? per user? per task?
  - path must be able to be created by XMLSERVICE job and must not be journaled, /tmp good place for it
- Set IPC path with ipc option to iLibCall, iDB2Call, iRestCall

Advanced Toolkit

- Scenario: RPG program to get back a list of objects that haven't been saved
- Two output parameters
  - NumObjs – number of objects returned in second parm (10i0)
  - Objects – array of 1000 object structures
    - Name – object name (10a)
    - Library – object library (10a)

Advanced Toolkit

```
itool.add(iPgm('unsaved', 'QUNSAVED')
 .addParm(iData('NumObjs','10i0',''))
 .addParm(
  iDS('Objects', {'dim':'1000'})
  .addData(iData('Name','10a',''))
  .addData(iData('Library','10a','')))

data = itool.dict_out('unsaved')
print(len(data['Objects']))
for obj in data['Objects']
    print(obj)
```

Advanced Toolkit

```
1000
{'Name': 'OBJ001', 'Library': 'QGPL'}
{'Name': 'OBJ002', 'Library': 'QGPL'}
{'Name': 'OBJ003', 'Library': 'QGPL'}
{'Name': 'OBJ004', 'Library': 'QGPL'}
{'Name': '', 'Library': ''}
...
{'Name': '', 'Library': ''}
{'Name': '', 'Library': ''}
```

Advanced Toolkit

```
itool.add(iPgm('unsaved', 'QUNSAVED')
 .addParm(iData('NumObjs','10i0',''))
 .addParm(
  iDS('Objects', {'dim':'1000'})
  .addData(iData('Name','10a',''))
  .addData(iData('Library','10a','')))

data = itool.dict_out('unsaved')
num_objs = data['NumObjs']
print(num_objs)
for obj in data['Objects'][:num_objs]
    print(obj)
```

Advanced Toolkit

```
4
{'Name': 'OBJ001', 'Library': 'QGPL'}
{'Name': 'OBJ002', 'Library': 'QGPL'}
{'Name': 'OBJ003', 'Library': 'QGPL'}
{'Name': 'OBJ004', 'Library': 'QGPL'}
```

Advanced Toolkit

```
itool.add(iPgm('unsaved', 'QUNSAVED')
 .addParm(iData('NumObjs', '10i0', '',
{'enddo':'count'}))
 .addParm(
  iDS('Objects', {'dim':'1000', 'dou':'count'})
  .addData(iData('Name','10a',''))
  .addData(iData('Library','10a','')))
data = itool.dict_out('unsaved')
print(len(data['Objects']))
for obj in data['Objects']
    print(obj)
```

Advanced Toolkit

```
4
{'Name': 'OBJ001', 'Library': 'QGPL'}
{'Name': 'OBJ002', 'Library': 'QGPL'}
{'Name': 'OBJ003', 'Library': 'QGPL'}
{'Name': 'OBJ004', 'Library': 'QGPL'}
```

Advanced Toolkit

- Scenario: System API call that takes in a large buffer with variable sized output data and needs to know the size of the buffer passed in
- One output parameter
  - Buffer – API defined data structure with a header containing numerous fields, followed by an array of data structures
- One input parameter
  - Len – size of Buffer

Advanced Toolkit

```
itool.add(iPgm('apicall', 'QAPICALL')
 .addParm(
  iDS('API0001', {'len':'buflen'})
  .addData(iData('Fld1','4b',''))
  # ...
  .addData(iData('NumResults','10i0','',
{'enddo':'count'}))
  .addData(iDS('results', {'dim':'1000',
'dou':'count'})
    .addData(iData('Res1','3i0','')))
    # ...
 .addParm(iData('BufSize','10i0','',
{'setlen':'buflen'})))
```

Advanced Toolkit

- Scenario: Program takes two varying length character strings
- Varying character strings not supported natively by itoolkit, need to use iXML

```
# XMLSERVICE/ZZSRV.ZZVARY:
#     P zzvary          B                   export
#     D zzvary          PI            20A   varying
#     D  myName                       10A   varying
```

Advanced Toolkit

```
itool.add(iXml("""
<pgm name='ZZSRV' func='ZZVARY' var='zzvary'>
  <parm io='in'><data var='myNameIn' type='10A'
varying='on'><![CDATA[<Ranger>]]></data></parm>

  <return><data var='myNameOut' type='20A'
 varying='on'><![CDATA[<Mud>]]></data></return>
</pgm>"""))

itool.call(config.itransport)
zzvary = itool.dict_out('zzvary')
if 'success' in zzvary:
  print(zzvary['myNameOut'])
```

Questions?

Power Systems Social Media

IBM Power Systems Official Channels:

https://facebook.com/IBMPowerSystems

https://twitter.com/IBMPowerSystems

https://www.linkedin.com/company/ibm-power-systems

http://www.youtube.com/c/ibmpowersystems

https://www.ibm.com/blogs/systems/topics/servers/power-systems/

More to Follow:

Blogs	Twitter	#Hashtags
IBM Systems Magazine You and i (Steve Will) IBM Systems Magazine i-Can (Dawn May) IBM Systems Magazine: iDevelop (Jon Paris and Susan Gantner) IBM Systems Magazine: iTalk with Tuohy IBM Systems Magazine: Open your i (Jesse Gorzinski) Trevor Perry Blog IBM DB2 for i (Mike Cain) IBM DB2 Web Query for i (Doug Mack) Modern-i-zation (Tim Rowe)	@IBMSystems @COMMONug @IBMChampions @IBMSystemsISVs @LinuxIBMMag @OpenPOWERorg @AIXMag @IBMiMag @ITJungleNews @SAPonIBMi @SiDforIBMi @IBMAIXeSupp @IBMAIXdoc	#PowerSystems #IBMi #IBMAIX #POWER8 #LinuxonPower #OpenPOWER #HANAonPower #ITinfrastructure #OpenSource #HybridCloud #BigData #IBMiOSS

Special notices

This document was developed for IBM offerings in the United States as of the date of publication. IBM may not make these offerings available in other countries, and the information is subject to change without notice. Consult your local IBM business contact for information on the IBM offerings available in your area.
Information in this document concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquires, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.
All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
The information contained in this document has not been submitted to any formal IBM test and is provided "AS IS" with no warranties or guarantees either expressed or implied.
All examples cited or described in this document are presented as illustrations of the manner in which some IBM products can be used and the results that may be achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions.
IBM Global Financing offerings are provided through IBM Credit Corporation in the United States and other IBM subsidiaries and divisions worldwide to qualified commercial and government clients. Rates are based on a client's credit rating, financing terms, offering type, equipment type and options, and may vary by country. Other restrictions may apply. Rates and offerings are subject to change, extension or withdrawal without notice.
IBM is not responsible for printing errors in this document that result in pricing or information inaccuracies.
All prices shown are IBM's United States suggested list prices and are subject to change without notice; reseller prices may vary.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this document may have been estimated through extrapolation. Users of this document should verify the applicable data for their specific environment.

Revised September 26, 2006

Special notices (cont.)

IBM, the IBM logo, ibm.com AIX, AIX (logo), AIX 5L, AIX 6 (logo), AS/400, BladeCenter, Blue Gene, ClusterProven, DB2, ESCON, i5/OS, i5/OS (logo), IBM Business Partner (logo), IntelliStation, LoadLeveler, Lotus, Lotus Notes, Notes, Operating System/400, OS/400, PartnerLink, PartnerWorld, PowerPC, pSeries, Rational, RISC System/6000, RS/6000, THINK, Tivoli, Tivoli (logo), Tivoli Management Environment, WebSphere, xSeries, z/OS, zSeries, Active Memory, Balanced Warehouse, CacheFlow, Cool Blue, IBM Systems Director VMControl, pureScale, TurboCore, Chiphopper, Cloudscape, DB2 Universal Database, DS4000, DS6000, DS8000, EnergyScale, Enterprise Workload Manager, General Parallel File System, , GPFS, HACMP, HACMP/6000, HASM, IBM Systems Director Active Energy Manager, iSeries, Micro-Partitioning, POWER, PowerExecutive, PowerVM, PowerVM (logo), PowerHA, Power Architecture, Power Everywhere, Power Family, POWER Hypervisor, Power Systems, Power Systems (logo), Power Systems Software, Power Systems Software (logo), POWER2, POWER3, POWER4, POWER4+, POWER5, POWER5+, POWER6, POWER6+, POWER7, System i, System p, System p5, System Storage, System z, TME 10, Workload Partitions Manager and X-Architecture are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries.

A full list of U.S. trademarks owned by IBM may be found at: http://www.**ibm.com**/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
AltiVec is a trademark of Freescale Semiconductor, Inc.
AMD Opteron is a trademark of Advanced Micro Devices, Inc.
InfiniBand, InfiniBand Trade Association and the InfiniBand design marks are trademarks and/or service marks of the InfiniBand Trade Association.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.
Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries or both.
Microsoft, Windows and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries or both.
NetBench is a registered trademark of Ziff Davis Media in the United States, other countries or both.
SPECint, SPECfp, SPECjbb, SPECweb, SPECjAppServer, SPEC OMP, SPECviewperf, SPECapc, SPEChpc, SPECjvm, SPECmail, SPECimap and SPECsfs are trademarks of the Standard Performance Evaluation Corp (SPEC).
The Power Architecture and Power.org wordmarks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.
TPC-C and TPC-H are trademarks of the Transaction Performance Processing Council (TPPC).
UNIX is a registered trademark of The Open Group in the United States, other countries or both.

Other company, product and service names may be trademarks or service marks of others.

Revised December 2, 2010