1. Introduction

This SunShell program communicates with the TTX unit via the RS232 port. It uses ITLA commands as specified in the OIF-ITLA-MSA document. It also include additional commands for lower level communications.

2. Shell description

SunShell is an instance of a Python shell. All programming features in Python version 2.2.2 may be used within SunShell. However, Python installation is not required to use SunShell. Visit the Python web site at http://www.python.org for documentations on Python programming.

3. Operation description

This section describes various levels of abstraction to communicate with the unit. An instance of ITLA object is instantiated within the shell. First, a connection must be established by the following command.

**Syntax:**
```
it.connect(com_port_number, baudrate)
```

**Example:**
```
it.connect(1, 9600) # connects to COM1 with baud rate 9600
```

3.1 Abstracted registers

Each ITLA command is abstracted by the name as it appears in the ITLA document. Type 'it.' to bring up an auto completion pull down list, which showing all supported commands. Each command will have a brief description of what it does and the return type.

3.2 Command exceptions
Note that commands below do not exactly match the behavior as described in the ITLA document for convenient purposes. To send it register alone, see section 3.4 Generic registers.

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>it.devTyp()</td>
<td>0x01</td>
<td>aea string of device type</td>
</tr>
<tr>
<td>it.mfgr()</td>
<td>0x02</td>
<td>aea string to manufacturer</td>
</tr>
<tr>
<td>it.model()</td>
<td>0x03</td>
<td>aea string of model</td>
</tr>
<tr>
<td>it.serNo()</td>
<td>0x04</td>
<td>aea string of serial number</td>
</tr>
<tr>
<td>it.mfgDate()</td>
<td>0x05</td>
<td>aea string of manufactured date</td>
</tr>
<tr>
<td>it.release()</td>
<td>0x06</td>
<td>aea string of release</td>
</tr>
<tr>
<td>it.relBack()</td>
<td>0x07</td>
<td>aea string of release backwards compatibility</td>
</tr>
</tbody>
</table>

### 3.3 Command modes read and write

The read mode is by default when no argument is given to a command. Otherwise the command mode is set to write. For example, `it.channel()` is in read mode to query the channel. And `it.channel(8)` is in write mode to tune to channel 8.

Read only commands do not have an argument list. To write to a read only command, one may construct a register by first reading it from the unit, modify the data, then issue a generic command `it.register` to write to the unit. See section 3.4 for other way to construct a register. Example,

```python
status, aeaEar = it.aeaEar()  # Get the register object
aeaEar.data(0xabcd)          # Update the field
it.register(aeaEar, 1)       # Write to unit
```

### 3.4 Generic registers

A generic (3 byte) register (1 byte address and two byte data) can be constructed by creating a Register object. The proper checksum will be automatically calculated and set before sending the register to the unit. Example,

```python
r = Register(address = 0xAB, data = 0xCDEF)
it.register(r)               # Read register from unit
it.register(r, 1)            # Write register to unit
```

### 3.5 Generic packets

A generic (4 byte) packet can constructed by creating a Packet object. One may set any field in the packet and send to the unit. Example,
mp = ModuleBoundPacket()  # Create packet for unit
mp.register(0xab)  # Update some fields
mp.checksum(0xf)
mp.data(0xface)
it.packet(mp, computeChecksum = 0)  # Send packet as is
it.packet(mp)  # Update checksum field and send

3.6 Generic byte streams

A generic byte stream buffer can be sent to and read from the unit. The buffer can be constructed as packet or register for convenient analysis. Example,

    it.write('abcd')  # Send 4 bytes to unit
    buffer = t.read(4)  # Read 4 bytes back

    hp = HostBoundPacket()  # Create host bound packet for analysis
    hp.buffer(buffer)  # Assign the raw buffer received
    print hp  # Inspect each field in packet

    # Create register using values from packet
    r = Register(hp.register(), hp.data())
    print r