Open Source WSJT
Status, Capabilities, and Future Evolution

Joe Taylor, K1JT

12th International EME Conference
Würzburg, August 25–27, 2006

Introduction

The software program WSJT originated in 2001 as part of an attempt to find optimum or near-optimum coding and modulation schemes for weak signal communication on the amateur VHF/UHF bands. My initial experiments were made with phase shift keying (PSK) on meteor scatter paths, but the first practical system made available to others used continuous-phase, 4-tone frequency shift keying (4-FSK) at 441 baud with noncoherent demodulation and a unique self-synchronizing method. This mode, called FSK441, is now the predominant one used for amateur meteor scatter communication throughout the world. A detailed description of it was published in the December 2001 issue of *QST*.[1]

In late 2001 I began experimenting with a 44-tone FSK system keyed at 5.38 baud, designed primarily for EME. A version of WSJT including the JT44 protocol was released in early 2002, and the first EME contacts with it were made soon afterward. A description of JT44 appeared in the June 2002 issue of *QST*.[2] A few months later the JT6M mode was introduced, using 44-FSK at 21.5 baud—a keying rate well suited to working ionospheric scatter on the 50 MHz band. The JT44 and JT6M modes worked well, but some early users more knowledgeable than I about digital communication techniques suggested that forward error-correction (FEC) might be beneficial, especially on the EME path. I was intrigued, but badly in need of education. Some months of study and many questions of experts were required before I could usefully proceed in this direction.

By late 2003 a new mode called JT65 had been designed and tested successfully on both terrestrial and EME paths. JT65 is a departure from previous WSJT modes in that it uses structured messages to allow lossless compression of callsigns and grid locators. With such compression, FEC enhancement becomes a necessity. The low-rate Reed Solomon code selected for JT65 makes a good match for the chosen FSK modulation, which consists of 64

[1] J. Taylor, K1JT, "WSJT: New Software for VHF Meteor-Scatter Communication," *QST* December 2001, pp. 36–41.

[2] J. Taylor, K1JT, "JT44: New Digital Mode for Weak Signals," *QST* June 2002, pp. 81–82.

orthogonal data tones and a 65th tone used for time and frequency synchronization. A state-of-the-art, award-winning "algebraic soft decision" algorithm[3] decodes received messages, and message averaging permits the decoding of messages too weak to be copied in a single transmission.

Several features were added to the JT65 decoder during its first year, including automatic frequency control to correct drifting signals and a hinted decoding procedure that takes advantage of a callsign database to suggest hypothetical messages to be compared against received data. Laboratory simulations were carried out to measure JT65's performance and reliability, and these measurements were confirmed by on-the-air experience. A user-level description of JT65 was published in the June 2005 issue of *QST,*[4] and full technical details of the JT65 protocol and its implementation within WSJT published a few months later in *QEX*.[5] JT65 has become popular because it offers an opportunity for even modest-sized stations (single yagis and 150 Watts) to make dozens of unique EME QSOs at 144 MHz.

Open Source

Recognizing that I am not getting any younger, and hoping that the considerable effort put into WSJT modes might evolve into even better things for amateur radio, in 2005 I undertook a major re-write of the program so that it could be made "open source." This goal was accomplished by the end of 2005; the program's source code, written in a combination of Python, Fortran, and C, is now available to anyone[6] under the terms of the GNU General Public License.[7] Working with the benefit of an open repository for the code, a core group of highly competent programmers are now actively engaged as the WSJT development team.[8] Sharing their valuable know-how, these individuals have compiled, operated, and made enhancements to WSJT on the Windows, Linux, and FreeBSD platforms. More than 200 other people have downloaded and studied the WSJT source code, as well, and many have offered valuable advice and counsel to the development team. The team welcomes new members, and there are many opportunities for good ideas in coding theory, digital signal processing,

[3]R. Koetter and A. Vardy, "Soft-Decision Algebraic Decoding of Reed Solomon Codes," in *IEEE Transactions on Information Theory,* **49**, 2809–2825, 2003.

[4]J. Taylor, K1JT, "EME with JT65," *QST*, June 2005, pp. 80–82.

[5]J. Taylor, K1JT, "The JT65 Communications Protocol", *QEX*, September-October 2005, pp. 3–12.

[6]Source code can be downloaded from `http://developer.berlios.de/projects/wsjt/`.

[7]Text of the GPL may be found at `http://www.gnu.org/copyleft/gpl.html`

[8]Team members presently include DL3LST, K1JT, KK7KA, N4HY, OH6EH, ON/G4KLX, VA3DB, and James Courtier-Dutton.

user-interface design, and the like. Most of us are amateurs in some or all of these areas, and we have much to learn! For those interested in programming details, an Appendix to this paper provides a structural overview of the WSJT program as of version 5.9.

Present Status

Unlike commercially engineered hardware or software, WSJT is a just-for-fun activity that has been pursued for its educational benefits and a desire to push forward the amateur weak-signal frontier. Part of the fun, and large portions of the educational benefits, are closely related to the program's status as a work always in progress. New ideas are frequently tried and tested, and when successful they may lead to new program features. I expect that this trend will continue and perhaps even accelerate as new people become involved.

Despite continuing program evolution, the basic capabilities of WSJT have been essentially stable since 2003 for the FSK441 and JT6M modes and since early 2005 for JT65. It appears that JT65 has become the mode of choice for the majority of amateur VHF EME contacts, except perhaps during CW-only contests or activity weekends. The reasons are easy to understand, and can be summarized in the following table summarizing minimum usable signal levels for various classes of JT65 messages and decodings:

Message type	Minimum S/N (dB)
Arbitrary message, including plain text . . .	−24
Callsign in database (Deep Search)	−28
Arbitrary message, with averaging	−29
Message synchronization	−30
Shorthand RO, RRR, 73	−32

The tabulated signal levels are quoted in dB relative to the noise power in a nominal SSB transceiver's 2500 Hz bandwidth. They measurements are based on numerical simulations using the JT65B submode, with simulated Rayleigh fading and controlled amounts of additive white gaussian noise. Additional details of the simulation results are contained in Figure 1.

For comparison, it is useful to note that signals begin to become audible around −16 dB on the WSJT scale (or +1 dB in 50 Hz bandwidth), depending somewhat on the details of receiver and ear-brain filtering. When copying messages of similar complexity, JT65 generally outperforms human-copied Morse code by 10–15 dB.

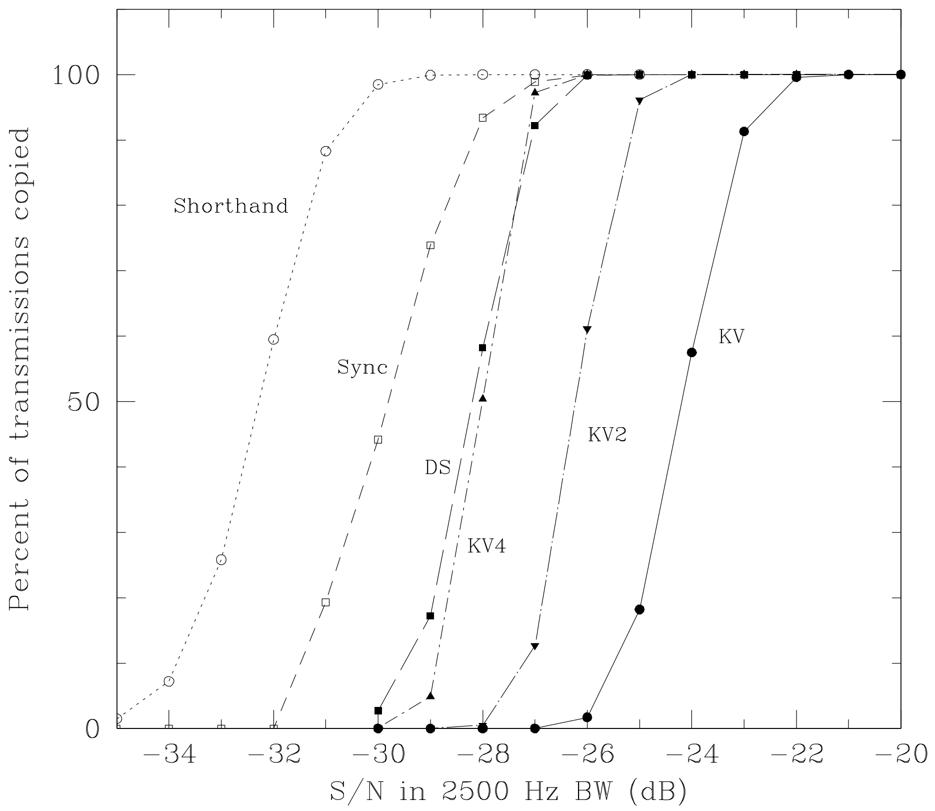

Fig. 1.— Percentage of simulated JT65B transmissions copied correctly, as a function of signal-to-noise ratio in 2500 Hz bandwidth. KV indicates Koetter-Vardy algorithm, while KV2 and KV4 indicate Koetter-Vardy decoding of the averages of 2 and 4 transmissions, respectively. DS indicates the Deep Search algorithm, "Sync" indicates achievement of time and frequency synchronization, and "Shorthand" indicates copy of the special messages RO, RRR, and 73.

JT65 has some characteristics that require getting used to. Unlike the more traditional coding and modulation methods widely employed in amateur weak-signal work, the reliability of copy with JT65 is not incrementally degraded as signals approach the minimum usable levels. Instead, one normally sees a whole message decoded correctly, or not at all. When hinted decoding (the so-called Deep Search decoder) is used, only hypothetical messages combining CQ or the user's own callsign with a callsign and locator from the callsign database will be tested against the received information. A stranger callsign—one not in the database—cannot be copied this way, nor can any plain text message. Such messages must be decoded by the general purpose Koetter-Vardy decoder, using message averaging if necessary.

The JT65 protocol is reasonably robust in the presence of low-level interference such as weak birdies. The decoders in WSJT cope well when several JT65 signals are present simultaneously, often allowing the operator to copy each signal separately even if they overlap in time and frequency.[9] The protocol's shorthand messages are an extremely effective and reliable way of exchanging minimal signal reports, acknowledgments, and end-of-QSO indications (RO, RRR, 73). Other message formats are available to convey multi-valued signal reports, suitably tagged with callsigns, whenever circumstances make them desirable.

During the development of WSJT, and especially while working on the EME-capable modes JT44 and JT65, I have enjoyed and benefited from a wide range of input from others. Dozens of experienced EME practitioners have been approached for advice and have willingly shared their views; excellent suggestions have been volunteered from an even larger number. Most recently the enlarged WSJT development team has contributed very significantly, as well. Inevitably, we have not satisfied all requests, but the evident popularity of JT65 seems to indicate that a good balance of features and capabilities has been achieved. Operating experience with JT65 now extends over some 2.5 years, and with that experience a heavy dependence on scheduled QSOs is rapidly fading away. As everyone at this conference knows, the thrill of an unexpected response to one's CQ directed at the moon can be a captivating experience.

Not surprisingly, some operators who are more comfortable with traditional coding and modulation techniques have stayed with the modes they like best—and that is as it should be. Morse code with on-off keying remains an extremely powerful and capable communication protocol, one that has much greater flexibility and a wider range of utility than the special-purpose protocols of WSJT.

Looking Ahead

I have mentioned that WSJT is a work in progress. What lies ahead? What might be the most telling motivations for future changes or additions? What about other possible implementations of the various signaling protocols developed for WSJT—perhaps in completely independent programs, or perhaps in new programs that might begin with some or all of the code base of WSJT, but then evolve quite separately? "Open source" means that the program code is freely available to anyone, and may be used or modified for any purpose, as long as the same licensing conditions are passed on to all users. My personal view is that these possible courses of future development are all highly desirable, and would bring good things to our hobby in this information age. I hope that at least some of them will come to pass.

[9] All JT65 transmissions are timed to start on a UTC minute; the JT65B signal bandwidth is about 355 Hz.

Some good ideas for future development are already being discussed. Two members of the present WSJT development team (KK7KA and ON/G4KLX) have some clever ideas about protocols that could be especially effective for microwave EME. Another member (N4HY) is working on an open-source version of a soft-decision Reed Solomon decoder, one that might outperform the Koetter-Vardy algorithm now used in WSJT. I have made early tests of a meteor-scatter mode that shows promise of significant advantages over FSK441, while retaining its desirable features, and also of a potential enhancement to JT65. I will comment only briefly on the latter ideas in this written paper; with good luck, some on-the-air performance measurements will be available in time for oral presentation at the Würzburg conference.

During the original design of JT65 I had in mind its possible use in terrestrial VHF/UHF contests. I also thought it might be desirable for standard JT65 messages to include QSO information beyond the minimal exchanges of callsigns and single-valued signal reports. After discussion with many others I settled on including the grid locator of the transmitting station in the basic message structure and allowing for multi-valued numerical signal reports. I also decided to place the majority of necessary information for a minimal valid QSO in the first exchanged messages—the ones containing callsigns, grid locators, and default OOO signal reports.

The exchange of grid locators is unnecessary, and not particularly useful in EME QSOs. They could reasonably be omitted. Further performance advantages might be gained by spreading the transmitted information more evenly over the procedural stages of a QSO. I outlined one way of doing this in a talk at the 11th International EME Conference, two years ago, and received favorable responses about it then and afterward. These ideas have now been developed further, and I am working on a possible superset of JT65 that would accommodate them in a backward-compatible way. An up-to-date progress report on that work will be presented at Würzburg.

Appendix: Programmer's Overview of WSJT

WSJT is a computer program designed to facilitate Amateur Radio communication under extreme weak-signal conditions. Three distinct coding and modulation methods are provided: FSK441 for communication by meteor-scatter techniques on the VHF bands; JT6M for meteor and ionospheric scatter, primarily on the 6 meter band; and JT65 for the very challenging Earth-Moon-Earth path. Each mode uses constant-envelope, phase-continuous FSK with keying rates and total bandwidths chosen to optimize effectiveness for the different propagation types, consistent with the limitations of typical amateur equipment.

WSJT runs under recent versions of the Windows, Linux, and FreeBSD operating systems. The program's user interface is written in the elegant and freely available language called Python. The Python code runs without changes on any supported platform, as long as the necessary modules are installed. The remaining source code can be compiled under Windows, Linux, FreeBSD, and Macintosh OS/X. Platform-dependent versions of FFTW, PortAudio, and libsamplerate need to be installed.

User Interface

The principal Python source-code files and their purposes include:

1. `wsjt.py:` Defines the main-screen graphical user interface (GUI) for user interaction; orchestrates all event-driven and time-shared activities.
2. `specjt.py:` Provides real-time display of received signals as two-dimensional "waterfall" spectra, with bandwidths and scrolling rates suitable for each mode.
3. `options.py:` Provides entry fields for user-defined parameters.
4. `astro.py:` Displays astronomical data for sun, moon, sky temperature, etc.

Several smaller Python files serve a number of minor utility functions.

Background Tasks

The main Python-coded procedures make calls to external routines compiled from code written in Fortran or C. A variety of global data is shared among modules through common blocks defined in Fortran. The Python code runs in a single thread, although timers make the functions of the several main modules appear concurrent. Fortran routines create additional threads used for soundcard I/O and the encoding and decoding of user messages.

As a small part of its overall task, the decoder for JT65 invokes an external program named KVASD (or KVASD.EXE) located in the main WSJT directory. If this program is present it uses information on received 64-FSK symbols and attempts to decipher it according to a Reed Solomon (63,12) code, using the algebraic soft-decision algorithm of Koetter and Vardy[3]. If KVASD is not present, WSJT uses its own internal hard-decision Reed Solomon decoder instead. Interprocess communication between WSJT and KVASD takes place through a shared disk file. KVASD is not an integral part of WSJT; its algorithm is patented, and the source code is the property of CodeVector Technologies, LLC. However, compiled versions of KVASD may be freely used in conjunction with WSJT for the purposes of amateur radio weak-signal communication.

Operation Sequence

WSJT execution starts at the top of Python file `wsjt.py`. The other Python modules are loaded and executed as needed. Fortran routines are called to start a high-priority thread that handles continuous A/D and D/A streams, as well as a background thread for decoding received or previously recorded signals. The top-level Python code determines the overall state of program operation, for example "Idle," "Monitoring," or "Transmitting." In normal usage the operator puts the program into "Auto" mode, which results in a timed sequence of alternating transmission and reception intervals.

Other Open-Source Software used in WSJT

WSJT 5.9 uses the following open source libraries:

1. FFTW, by Matteo Frigo and Steven Johnson, for computing Fourier transforms.
2. PortAudio, by Ross Bencina and Phil Burk, for audio I/O.
3. ALSA, "Advanced Linux Sound Architecture." For details, see http://www.alsa-project.org/.
4. Secret Rabbit Code or `libsamplerate`, by Erik de Castro, for accomplishing band-limited resampling of data.
5. RS, by Phil Karn, KA9Q, for Reed Solomon encoding and hard-decision decoding.

Partial List of Functions and Subroutines

In the following list of procedure names and descriptions, indentation gives a rough idea about the relative dependencies of functions.

```
Audio initialization, message encoding, and display computations:

   blanker.f90      Noise blanker
   fivehz.f90       Called by PortAudio callback at 5 Hz rate
   flat2.f          Flatten the spectrum for waterfall display
   pix2d65.f90      Compute pixels for waterfall display
   pix2d.f90        Compute pixels for waterfall display
   runqqq.f90       Execute another process
   wsjtgen.f90      Generate Tx waveforms
       abc441.f90   Part of FSK441 generator

     gen65.f        Generate JT65 waveform
       chkmsg.f     Check a JT65 message for presence of 'OOO'
       encode65.f   Encode a JT65 message
       getpfx1.f    Handle extra DXCC prefixes
       getpfx2.f    Handle extra DXCC prefixes
       graycode.f   Convert binary to/from Gray code
       grid2k.f     Convert grid locator to integer
       interleave63.f Interleave JT65 symbols
       nchar.f      Convert number, letter, space to 0-36
       packcall.f   Routines for JT65 source encoding
       packdxcc.f   Encode DXCC prefix
       packgrid.f   Encode grid locator
       packmsg.f    Assemble parts of a message
       packtext.f   Encode a plain text message
       pfx.f        List of add-on DXCC prefixes

     gen6m.f        Generate JT6M waveform
       gentone.f    Generate tone for JT6M message
     gencw.f        Generate CW waveform
       morse.f      Convert ascii to morse dits
     gencwid.f      Generate a CW ID message

   gcom1.f90        Global variables shared among Fortran, Python,
   gcom2.f90        and C routines
   gcom3.f90
   gcom4.f90

Astronomical calculations:

   astro.f          Computes Az, El, Doppler for Sun, Moon, etc.
   azdist.f         Computes azimuth, distance, etc., between two locators
   coord.f          Spherical trig utility
   dcoord.f         Spherical trig utility in double precision
   deg2grid.f       Convert lat/long (degrees) to grid locator
   dot.f            Compute dot product
   ftsky.f          Get sky temperature from data file
   geocentric.f     Convert geodetic to geocentric coords
   GeoDist.f        Compute azimuth and distance between two locators
   grid2deg.f       Convert grid locator to lat/long
   moon2.f          Compute moon location at specified date and time
   MoonDop.f        Compute lunar doppler shift and related quantities
   sun.f            Compute sun location at specified date and time
   toxyz.f          Convert from polar to cartesian coordinates

General Utilities:

   db.f             Compute decibels from ratio
   igray.f          Gray code
   indexx.f         Sort routine
   set.f            Move, add, zero, ...
   pctile.f         Sort an array and get specified percentile
   rfile2.f         Read a binary file
   sort.f           Sort an array
   cutil.c          Fortran wrappers for some basic C functions

 FFTs:

   fftw3.f          Fortran definitions for FFTW
   four2a.f         Wrapper to make FFTW look like four2
   four2.f          FFT in Fortran
   ps.f             Compute power spectrum
   xfft.f           Real to complex FFT wrapper
   xfft2.f          Real to complex FFT wrapper


Decoding Routines:

   wsjt1.f          Top-level decoding routine; handles FSK441 especially
     avesp2.f       Computes average spectrum
     bzap.f         Find and remove birdies
     detect.f       Measure power in FSK441 tones
     flatten.f      Flatten the spectrum
     longx.f        Decode normal FSK441 messages
     lpf1.f         Quick-and-dirty lowpass filter
     mtdecode.f     Multi-tone decoding
     ping.f         Find pings
     s2shape.f      Flatten the 2d spectrum
     smooth.f       Smooth by boxcar averaging
     spec2d.f       Compute 2d spectrum for FSK441
     stdecode.f     Decode FSK441 shorthand messages
     sync.f         Synchronize FSK441 data

   wsjt65.f         JT65 decoder
     afc65.f        AFC for JT65
     avemsg65.f     Decode average message
     decode65.f     Decode JT65 message
     deep65.f       Deep search decoder
     demod64a.f     Compute probabilities of transmitted symbols
     extract.f      Extract message from JT65 symbol probabilities
     flat1.f        Flatten the passband
     getsnr.f       Compute snr or shorthand message
     k2grid.f       Convert integer to 4-digit grid locator
     limit.f        Clipper for JT65
     peakup.f       Interpolate to find fractional-bin peak
```

setup65.f	Initialize pseudorandom sync vector
short65.f	Detect JT65 shorthand messages
slope.f	Remove a straight-line slope
spec2d65.f	Compute 2d spectrum for JT65
spec441.f	Compute spectra for FSK441 decoding
sync65.f	Synchronize a JT65 signal
unpackcall.f	Unpack callsign
unpackgrid.f	Unpack grid locator
unpackmsg.f	Unpack a JT65 message
unpacktext.f	Unpack a plain text message
xcor.f	Compute cross-correlation for JT65 sync

decode6m.f	Decode a JT6M signal
syncf0.f	First frequency sync
syncf1.f	Second freq sync
synct.f	First time sync
avemsg6m.f	Get average JT65 message

JT65code.f	Example program to illustrate and test JT65 coding and decoding

Hard-Decision Reed Solomon encoder and decoder:

decode_rs.c	Karn's RS decoder
encode_rs.c	Karn's RS encoder
init_rs.c	Initialization routine
wrapkarn.c	Wapper for Fortran

C Functions for Audio and T/R switching:

jtaudio.c	Audio I/O, calls PortAudio routines
padevsub.c	Select desired audio device
ptt.c	PTT via serial port DTR/RTS
ptt_linux.c	PTT via serial or parallel ports (*nix)
resample.c	Wrapper for resample routine
start_alsa.c	Start audio streams using ALSA
start_threads.c	Start audio and decoder threads