Special thanks to Jeff Coleman for suggesting this.

Consider the following account code:

Copy input data to memory bytes 32...

~calldatacopy(0, 32, ~calldatasize())

Copy sighash to after the end of the input bytes

~mstore(~msize(), ~sighash())

SHA3 memory bytes 128... (that is, input bytes 96... plus sighash), put result in bytes 0...31

~mstore(0, ~sha3(128, ~msize() - 128))

Check nonce (input bytes 96...127)

assert ~mload(128) == ~sload(0)

Verify signature

~ecrecover(3000, 1, 0, 0, 128, 0, 32) assert ~mload(0) == ~sload(1)

Pay for gas (input bytes 128...159)

~paygas(~mload(160))

Update nonce

~sstore(0, ~mload(128) + 1)

Self-delegatecall, using code from input bytes 160.... (not including sighash)

~delegatecall(-1, self, 192, ~msize() - 224, 0, 0)

Essentially, this does all of the usual signature and nonce checks and gas payment, but then instead of sending a single call to a specific address with a specific value and data, it simply runs a piece of code, which could make one or more function calls, issue logs, or perform any other operations. If the code does multiple things, it is guaranteed to do them atomically.

If the paygas rules allow it, a further simplification might be to also move nonce checking, nonce updating and gas payment into the code of each transaction, thereby allowing accounts to arbitrarily switch between anti-replay schemes and gas price policies if they so choose to.