江端さんのDIY奮闘記 EtherCATでホームセキュリティシステムを作る（4）：

ご主人様とメイドはテレパシー通信をしている？

http://eetimes.jp/ee/articles/1507/28/news022.html

さて、今回はEtherCATのメモリについてお話します。EtherCATのPDO通信、SDO通信は、ご主人様（マスタ）とメイド（スレーブ）は、このメモリを介した“テレパシー通信”によって、完璧なコミュニケーションを実現しています。それを説明した後に、いよいよ、本連載の山場の1つとして、SOEM（Simple Open EtherCAT Master）のデバッグ＆トレース環境の作り方を紹介します。

2015年07月29日 11時00分 更新

[江端智一, EE Times Japan]

FA（ファクトリオートメーション）を支える「EtherCAT」。この超高度なネットワークを、無謀にも個人の“ホームセキュリティシステム”向けに応用するプロジェクトに挑みます……!!⇒「江端さんのDIY奮闘記 EtherCATでホームセキュリティシステムを作る」連載一覧

前回、私は、「老人ホーム4.0」というコンセプトを紹介しました。

研究者も政府も当てにせず、―― 私の老後の人生のために、私の自宅の隅々に、私のために必要な介護アシスト機器を「自分で作り、自分だけのために動かす」―― ホームセキュリティシステムの作る、と宣言しました。

なぜ私が、「自分で作る（Do It Yourself：DIY）」ことにこだわっているかというと、市場原理やメーカーの経営戦略なんぞを当てにしていたら、100年たったとしても、私の考える「老人ホーム4.0」が決して実現されないことを、私は「知っている」からです。

「思っている」のではありません。「知っている」のです。

初版の原稿では、私は、この導入部に、何かが取りついたかのように、その「理由」を書きまくっていました。しかし、この原稿をレビューした後輩の乾いたひと言が、私を現実に引き戻しました。

―― EtherCATの連載なんですよね？ 江端さん、あなた、一体何やってるんですか？

われに返った私は、担当のMさんと相談して、その「理由」を全部、「付録」に回すことにしました。

しかし、私が、エンジニアであるあなたに本当に読んでいただきたいのは、この「付録」の方です。EtherCATの内容なんか、全部飛ばしたって構いません*）。

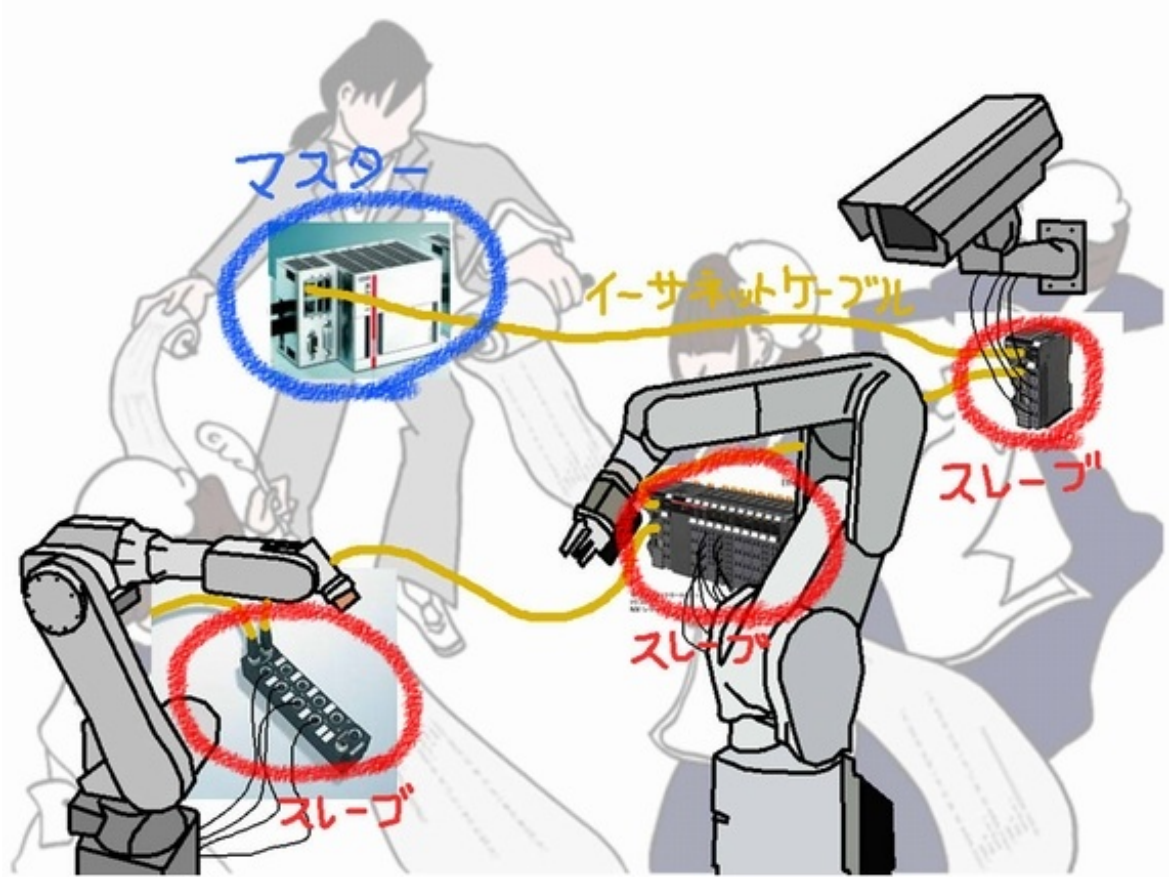
＊）江端さん。冗談でも、そーゆーことは書かないで下さい（担当M）

江端の企業エンジニアの半生の「憎悪」と「怨念」と「悲哀」、そして、私が「週末研究員」という概念を獲得するに至った経緯を、是非皆さんと共有したいのです。

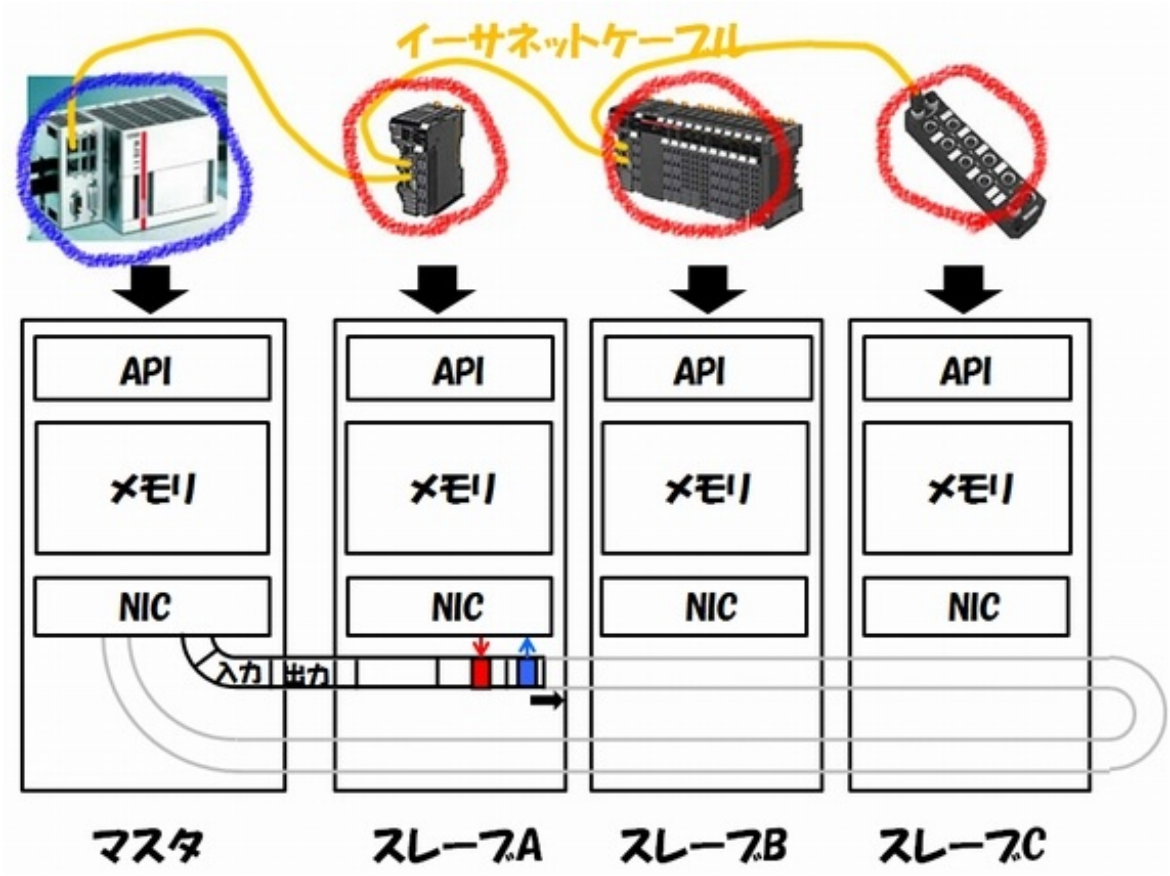
本文より文字数が多いという、EE Times Japan連載史上、前代未聞の「付録」。ぜひ、ご一読ください。

こんにちは、江端智一です。

前回は、EtherCATのSDO通信を使ったステータスとその管理方法について説明しました。今回は、EtherCATのざっくりとしたアーキテクチャと動作原理についてお話したいと思います（なお、今回も、EtherCATマスタ→ご主人様、EtherCATスレーブ→メイドで押し通します）。



結局のところ、EtherCATは、ご主人様もメイドも同じアーキテクチャからできています。ハードウェア構成はPCと同様に、アプリケーションインタフェース（API）と、記憶装置（メモリ）と、通信装置（ネットワークインターフェース：NIC）の3つです。

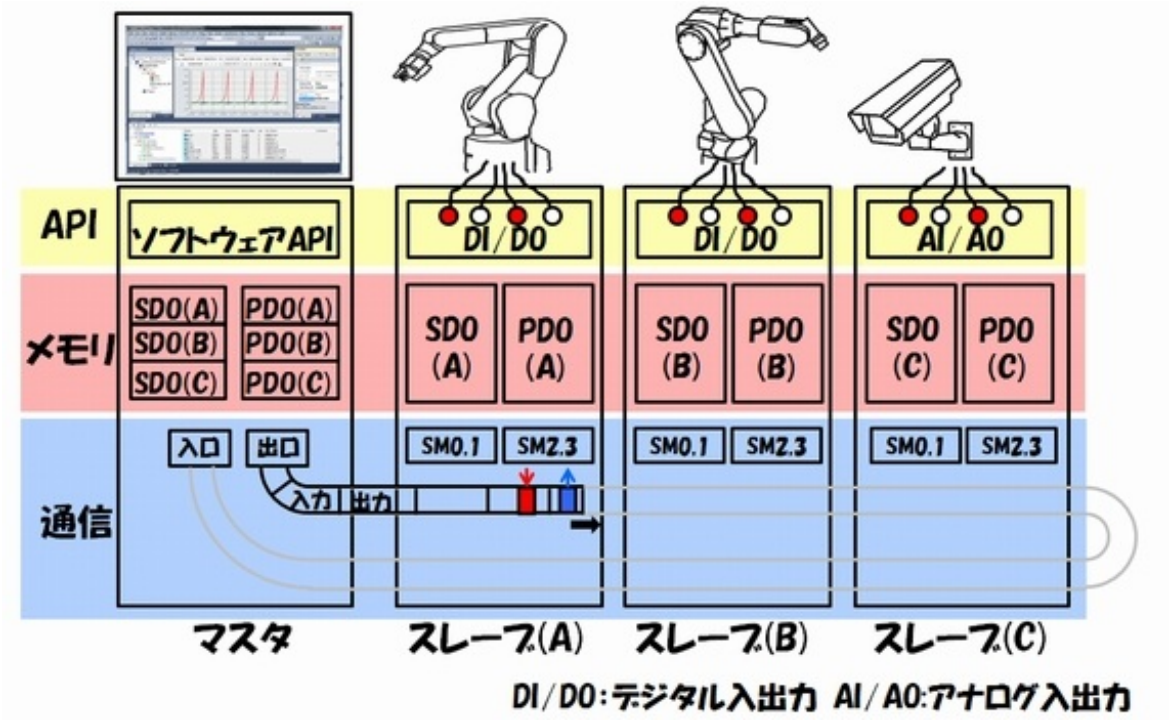


違いといえば、APIの上で動くアプリケーションが、エクセルやワードではなく、マスタ用の制御プログラム、スレーブが動かす介助ロボット、各種センサーであるところです。

さらに、EtherCATがPCと決定的に異なる点としては、上記のAPIとメモリとNICの3つが、お互いの存在を無視するかのようにバラバラに動いているという点です。

メモリに何が書き込まれようとも（あるいは、書き込まれなくても）、NICは、1秒間に最大8000回の通信を続け、APIは、介助ロボット、各種センサーが要求するタイミングで、制御に必要なデータ（ロボットアームの角度や動かす速度）を受け渡すだけです。

PCであれば、APIの上で動くアプリケーションが、メモリ、NICにそれぞれ指示を出します。しかし、EtherCATでは、ご主人様またはメイドが、分裂した3つの多重人格を持っていて、それぞれの人格が勝手に自分の仕事を黙々と続けているだけです。



「なんで、そんな効率悪いことやってんの？」と問われれば、制御システムのリアルタイム性と同期を確保するためなのです。

ご主人様（マスタ）とメイド（スレーブ）はぴったりと息を合わせなければ、全体として制御システム（製造ラインなど）を動かすことができません。もし、通信のタイミングがちょっとでもズレるようなことがあれば、たちまち、その制御システムは「止まる」か「暴走」してしまいます。

例えば、Webブラウザのように、表示されるまでの時間が事前に分かっていないようなプログラムでは、到底、制御システムでは使えないのです。

息ぴったり！ ご主人様とメイドたち

さて、今日は、このAPI、メモリ、通信の3つのうちの、メモリを中心にお話ししたいと思います。

まず、プロセスデータオブジェクト（Process Data Object：PDO）通信と、サービスデータオブジェクト（Service Data Object：SDO）通信のおさらいをします。

通信方式	概要	「ご主人様ーメイド」で説明すると
PDO通信	マスタが、スレーブに対して、デバイス（ロボット等）を動かす具体的な数値を送り込む	ご主人様が、メイドに対して、1秒間に100～1000回以上も「ああせい、こうせい」との指示を出す。
SDO通信	マスタがスレーブの情報（構成情報や、健康状態）を聞き出す	ご主人様が、メイドの出身地や特技、アイテムを聞き出し、または、ちゃんと働ける状態にあるかチェックする

……と、ここまで書いて、ふと思ったのですが、「PDOとかSDOとか一体何なの？」と思っている人っているんじゃないかなー、と思ったりしています（実際、私にはよく分かりませんでした）。

そこで、乱暴な理解ですが、「オブジェクト＝スレーブ装置の中のメモリ」と考えてしまいます。つまりPDO通信とは、PDO用のメモリを読み書きする通信であり、SDO通信とはSDO用のメモリを読み書きする通信である、と考えてしまえばいいのです。

PDO通信は、プロセスデータオブジェクト（PDO）用のメモリに、“1”や“0”を書き込むだけです。それらの制御データは、APIを介して、ロボットに転送される場合もありますし、されない場合もあります。ロボットに制御データが転送される前に、メモリのデータの内容が書き換えられてしまう場合もありますが、それでよいのです*）。

*）正確には3つのタイプの同期モードがあるのですが、今回はこれで押し通します。

一方、サービスデータオブジェクト（SDO）用のメモリには、スレーブの構成情報や健康状態が書き込まれていて、それを、ご主人様、前回説明したメールボックス通信で、回収するものと理解します。こちらも、読み書きのタイミングについては、PDO通信と同様に、同期は取りません。

メイドは、「どのAPIに向けて、どのようなタイミングで、どんな信号を出す」とか、「ご主人様に対してどのような通信メッセージを作って返信するか」などを、一切考える必要はなく、単にメモリに情報を書き込むだけで、ロボットやご主人様と連絡が取れてしまうのです。これはメイドにとって、とてもラクチンなはずです。

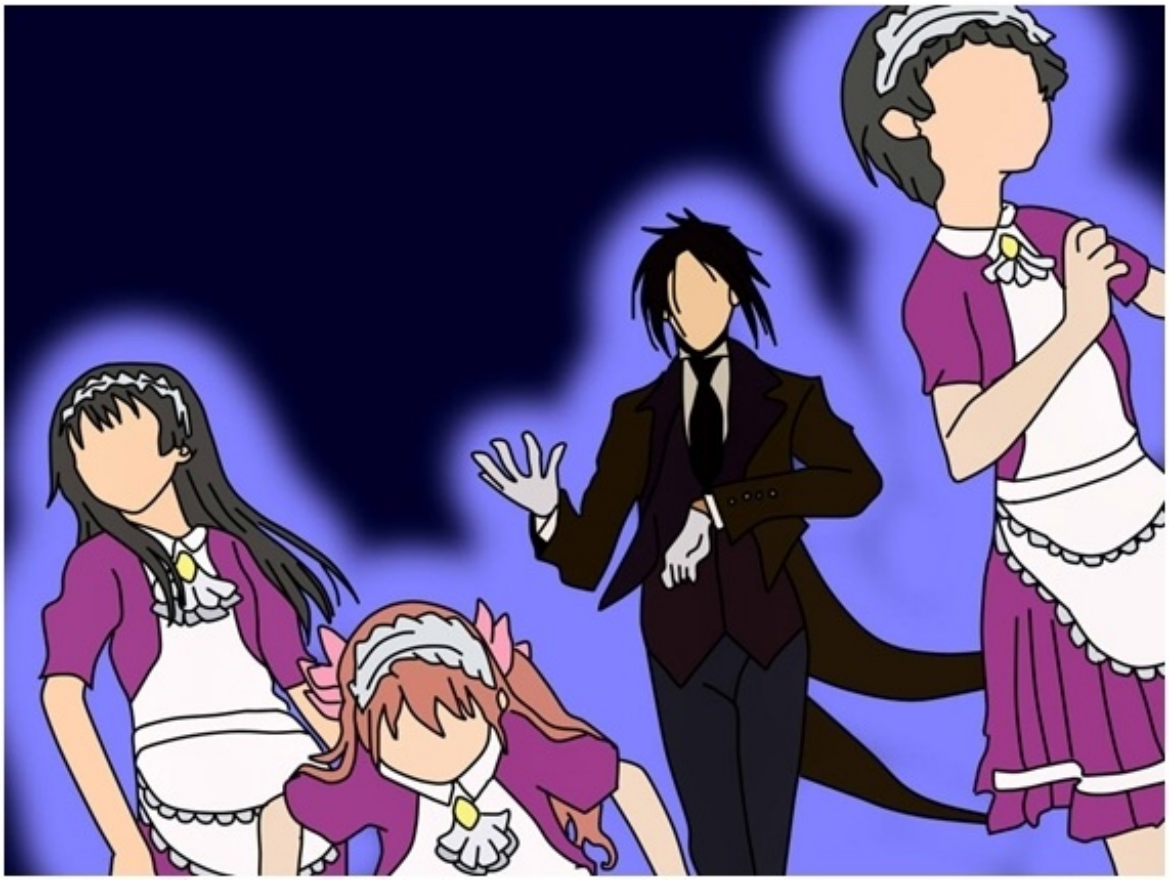
ご主人様は、さらにすごいです。

自分のメモリの中に、メイド全員のSDOとPDOの情報を持っていて（厳密にいうと、ちょっと違うのですが、今回はこれで押し通します）、ご主人様が、自分で持っているメイド用メモリの情報を書き替えるだけで、自動的にその情報がメイドのメモリの情報に反映されてしまうのです。

このすごさをどのように表現すればよいのでしょうか。

ご主人様が、メイドに「こうして欲しい」と考えるだけで、その気持ちがメイドに伝わり、メイドが「こうしました」と考えるだけで、ご主人様はそれを知ることができる。

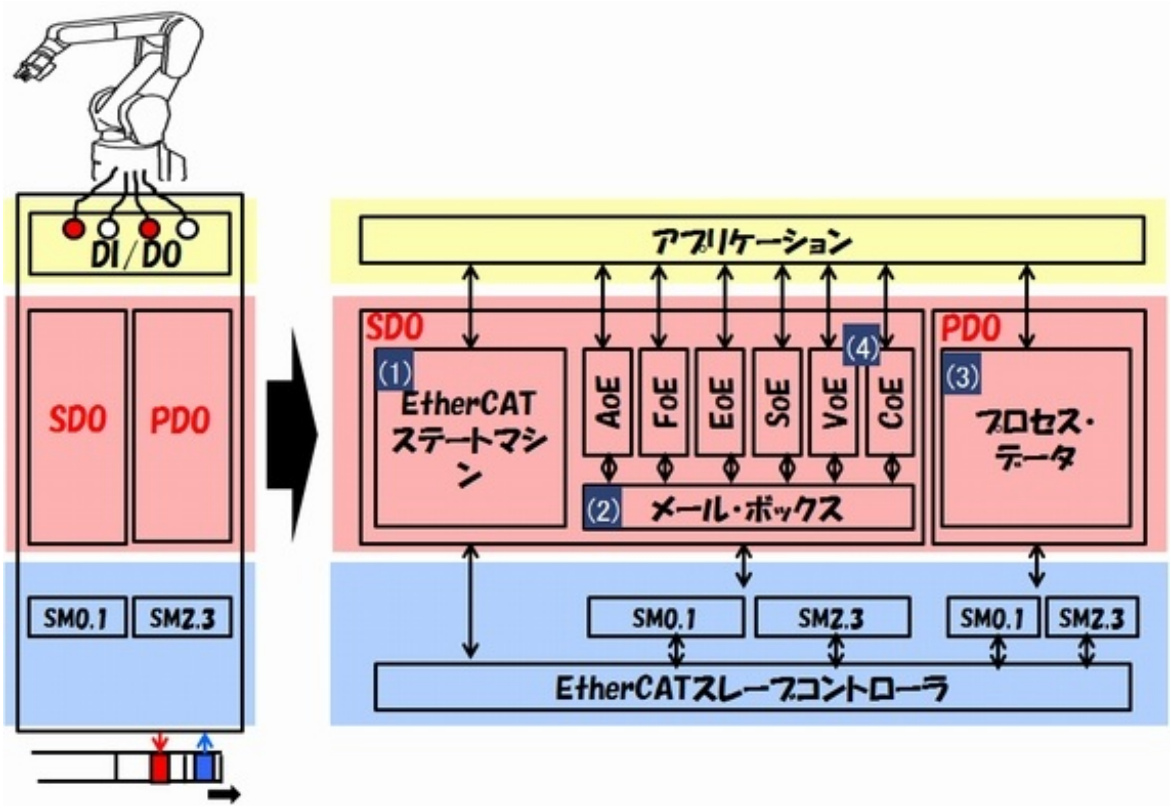
これはもうテレパシー通信と言っちゃってもいいんじゃないかなー、と思うのですよ。



ご主人様とメイドたちは、テレパシー通信ができる

各メモリを細かく見てみる

このEtherCATのPDO用のメモリと、SDO用のメモリをもう少し細かく見てみましょう。



まず、SDOの中にある「(1)ステートマシン」と「(2)メールボックス」通信については、前回お話した、『ご主人様とメイドのプライベートな手紙のやりとり』で実現されます。

「(3)プロセスデータ」とは、PDO用のメモリにロボットアームの角度や動かす速度などが読み書きされるデータのことです。1秒間に最大8000回のPDO通信と、イーサネットのNICとロボットと繋がったAPIの両方から読み書きされ続けます ―― と、ここまでは、前回までの復習です。

さて、最後の最後まで訳が分らなかったものが(私だけかもしれませんが)、(4)のAoEやらFoEやらCoEというモノでした。

例えば、CoEの説明で、

『広範囲のデバイスクラスとアプリケーションで利用でき、I/Oコンポーネント、ドライブ、エンコーダ、比例バルブや油圧コントローラなど、例えばプラスチックや繊維機械のアプリケーションプロファイルに対し使用されています』

―― うん、はっきり言って、私には、何が書かれているのかさっぱり分からん。

ですので、ここでは、現状の私の理解で押し通します(業界関係者の方のツッコミをお待ちします)。

まず、今は、CoE以外のことは全部忘れてしまいましょう。AoEやらFoEが登場してくることは(多分)ないと思いますので。

CoEとは、CAN application protocol over EtherCATの略称です。ここでCANとは、

Controller Area Networkのことで、車載ネットワークの事実上の標準です。ですから、“Car Area Network”と間違う人が多いのです（例えば「私」）。

CoEは、ひと言で言えば、スレーブまたはスレーブの管理するロボットの仕様書（マニュアル）です。要するに、定型フォーマットに従って、メイドが取り扱う機器（ロボットなど）の仕様を記載したもので、これをプロファイルと呼びます。エクセルのテンプレートに従って記載されたファイル、あるいは、お役所に提出する「出生届け」と同じものと言ってもよいでしょう。

EtherCATが登場するずっと前から、CAN用のプロファイルは沢山ありました。自動車はもちろん、産業用ロボット、医療機器、鉄道、ビルオートメーションなど、それはもう山ほどです。

プロファイルには、機器の名称や、製品番号、メーカー名などの他にも、入出力用のインタフェースや、そのデータ形式なども記載されることになります。

ここで、EtherCATの開発者は、このように考えたのかもしれません。

・EtherCAT用にプロファイルを、一から作るのって面倒だなぁ
・それなら、CANのプロファイルをそのままパクって……もとい、活用できるといいなぁ
・うまくいけば、CANを使っていた人も、簡単にEtherCATに乗り変えてもらえそうだなぁ

（あくまで江端の推測ですよ、推測）

このように、「EtherCATでCANのアプリケーションのプロファイルを使えるようにする」という内容を、英語にすれば、“CAN application protocol over EtherCAT”、すなわち、「CoE」になるわけです。

このスレーブの情報を、マスタが手に入れることによって、マスタはスレーブが管理している機器の動かし方が分かるようになり、前述した「テレパシー通信」で、ご主人様がメイドを直接コントロールできるようになるのです。

SOEMのデバッグ＆トレース環境の作り方

では最後に、連載第4回にして、ようやく本連載の大きな山の1つである、Visual C++ 2010（無償版）を使った、EtherCATマスタであるSOEM（Simple Open EtherCAT Master）のデバッグ＆トレース環境の作り方について説明致します。

第2回で、SOEMの実行ファイルの作り方については説明しましたが、この実行ファイルでは、SOEMが実際にどのように動いているかを知ることができません。

しかし、SOEMは、全てのソースコードが開示されていますのでVisual Studioのデバッグモードを用いれば、SOEMを動かしながらEtherCATマスタの動きを追いかけることができます。

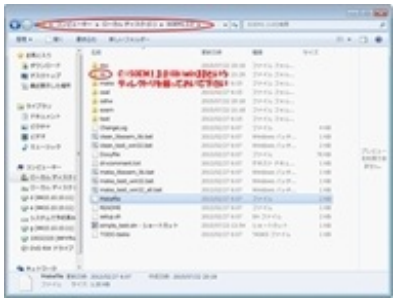
また、Wireshark（ネットワークアナライザ）などで、ネットワークをモニタすることで、送信フレームの内容も併せて知ることができるようにもなります。

では、説明を始めます（興味のない方は、「付録」のページまでスキップして下さい）

前提として、連載第2回のP.5に記載されている「SOEMでEtherCATを作ってみる」の「（1）準備していただくもの」から、P.7の「SOEMの構築」の「（Step.2）Visual C++ 2010（無償版）をインストールする」までを実施して頂けているものとします。

【Step 1】

SOEMのソースコードを解凍した状態です。



C:¥直下に展開してください（私の稼働環境と同じにしていただくためです）。C:¥SOEM1.3.0となります。また、C:¥SOEM1.3.0¥lib¥win32というディレクトリを掘っておいてください（SOEMのライブラリ（libsoem.lib）を、ここに作ります）。

【Step 2】

Microsoft Visual C++ 2010 expressを起動し、「ファイル（F）」→「新規作成（N）」→「既存のコードからプロジェクトを作成」、と選択すると、右の画面が出てきます。

【Step 3】

「次へ」を選択し、プロジェクトファイルの場所（L）:→c:¥SOEM1.3.0¥test¥win32¥simple_test→フォルダーの選択→プロジェクト名（R）:→simple_test

でいきなり「完了」ボタン。これでsimple_testのプロジェクトができます。

【Step 4】

（Step.2に戻って）同様に、

プロジェクトファイルの場所（L）:→C:¥SOEM1.3.0¥test¥win32¥slaveinfo→フォルダーの

選択→プロジェクト名（R）:→slaveinfoでいきなり「完了」ボタン。これでslaveinfoのプロジェクトができます。

【Step 5】

（Step.2に戻って）同様に、

プロジェクトファイルの場所（L）:→プロジェクトファイルの場所（L）
:→C:¥SOEM1.3.0¥soem→フォルダーの選択→プロジェクト名（R）:→soemでいきなり「完了」ボタン。これでsoemのプロジェクトができ、3つのプロジェクトの生成が完了します。

【Step 6】

3つのプロジェクトをバラバラに管理すると面倒なので、simple_testのプロジェクトに3つのプロジェクトを集めます。

一度、Microsoft Visual C++ 2010 Expressを落としてから、
C:¥SOEM1.3.0¥test¥win32¥simple_test¥ simple_test.slnをクリックして、simple_testのプロジェクトを立ち上げてください。

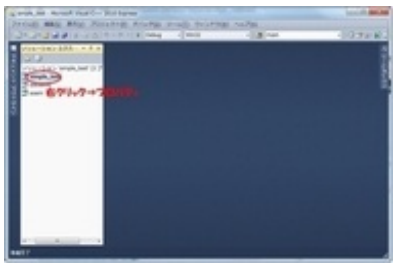
さらに、ファイル（F）→追加（D）→既存のプロジェクト（E）から、
C:¥SOEM1.3.0¥test¥win32¥slaveinfoの中にできている、slaveinfo.vcxprojを選んで、「開く」を押してください。

同様に、「ファイル（F）」→「追加（D）」→「既存のプロジェクト（E）」から、
C:¥SOEM1.3.0¥soemの中にある、soem.vcxprojを選んで「開く」を押してください。

これで、simple_test（簡易EtherCATテストプログラム）のプロジェクトで、slaveinfoプロジェクト（スレーブ情報収集プログラム）と、soemプロジェクト（SOEM）ライブラリのプロジェクトを同時に管理できるようになります。

【Step 7】

Simple_testの設定を開始します。



【Step 8】

「構成プロパティ」→「全般」として「アプリケーション（.exe）」、
「文字セット」→「マルチバイト文字セットを使用する」→「適

用（A）」

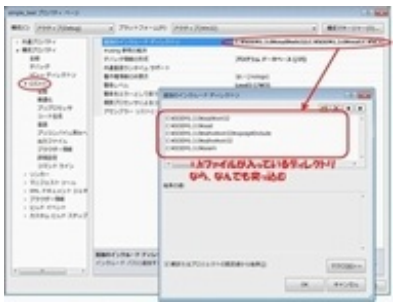
【Step 9】

「デバッグ」→「コマンド引数」→使用するNICアダプタを入力する→「適用（A）」（こちらを参照）

ちなみに、下図のNICは、あなたのPC環境のものを入力してください）

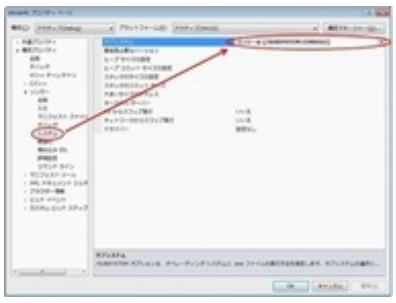
【Step 10】

「C/C++」→「追加のインクルードディレクトリ」に、5つのディレクトリ名を入力してください。→「適用（A）」



【Step 11】

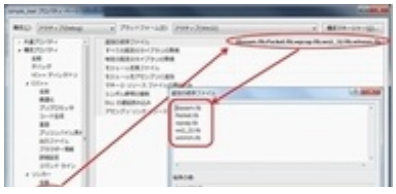
「リンカー」→「システム」→「サブシステム」が「コンソール（/SUBSYSTEM:CONSOLE）」になっていることを確認してください。



【Step 12】

「リンカー」→「入力」→「追加の依存ファイル」に、あなたの環境での下図の5つのファイル名を入力。

これで、プロジェクト”simple_test”についての設定は完了

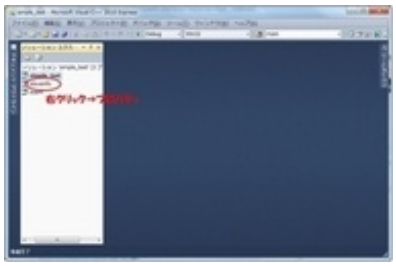


です。

【Step 13】

slaveinfoの設定を開始してください。

Step.8〜12と同じ設定を行ってください。



【Step 14】

soem（ライブラリ）に必要なファイルを追加してください。

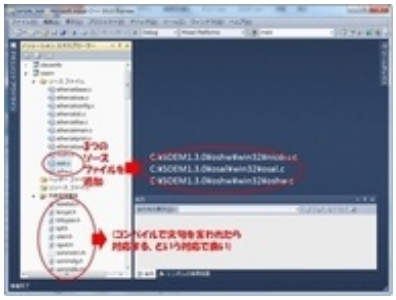
【Step 15】

soemに必要な、以下の3つのファイルを追加してください。

C:¥SOEM1.3.0¥oshw¥win32¥nicdrv.c

C:¥SOEM1.3.0¥osal¥win32¥osal.c

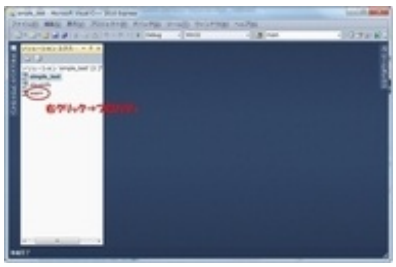
C:¥SOEM1.3.0¥oshw¥win32¥oshw.c



また、ヘッダファイルは、最初はほっといてもよいでしょう（コンパイルエラーが教えてくれるので、その時追加してもよいです）。

【Step 16】

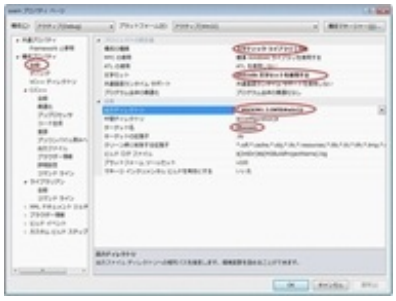
soemの設定を開始する。



【Step 17】

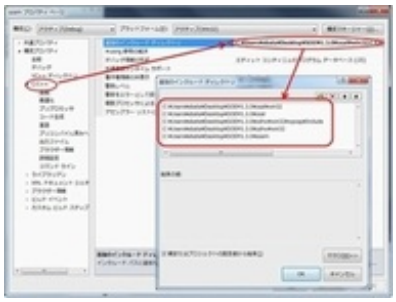
「構成プロパティ」→「全般」→「構成の種類」として「スタティックライブラリ（.lib）」、「文字セット」→「マルチバイト文字セットを使用する」、出力ディレクトリ→“C:¥SOEM1.3.0¥lib¥win32”

「ターゲット名」→“libsoem”



【Step 18】

「C/C++」→「全般」→「追加のインクルードディレクトリ」に、5つのディレクトリ名を入力



この設定が終了したら、最初にsoemプロジェクトのビルドを実施します。成功した後、slaveinfo, simple_testのプロジェクトでビルドしてください（ビルドの順番に注意してください）。これで、slaveinfo, simple_testのデバッグ＆トレース環境の構築が構築できていると思います。

この環境を立ち上げるときは、C:¥SOEM1.3.0¥test¥win32¥simple_test¥simple_test.slnを叩けば、3つのプロジェクトが同時に表示されます（simple_test.slnは、C:¥SOEM1.3.0あたりに、ショートカットを張っておくと便利です）。

もしうまく環境が構築できなかったらご連絡ください。メールベースで可能な限りサポート致します（なにしろ私の手元には、実際に動いているデバッグ＆トレース環境がありますし）。

ただ、私は、親切心からあなたをサポートをするわけではありません。

これは、私の計画「老人ホーム4.0」に、あなたを巻き込むための布石です。

来週からは、このデバッグ＆トレース環境環境を使って、EtherCATの動きを一つ一つ見ながら、理解を深めていきたいと思います。

では、今回のコラムの内容をまとめます。

(1)EtherCATは、APIとメモリとNICの3つが、お互いの存在を無視するかのようにバラバラに動き、その事によって、抜群のリアルタイム性能と同期性能を実現している
(2)EtherCATは、既存の制御ネットワークのデバイスプロファイルをパクる……もとい、活用することで、デバイスの相互利用性を担保している
(3)SOEM(Simple Open EtherCAT Master)のデバッグ&トレース環境の構築は、結構面倒くさい(―― ですが、なんとか頑張って構築してみてください)

特別協力:

本連載では、スレーブの提供などでベッコフオートメーションにご協力いただいております。

BECKHOFF

それでは、次ページから「付録」を開始します。準備はいいですか?

なぜ「DIY」にこだわるのか

前回、私は、「老人ホーム4.0」というコンセプトを紹介しました。

研究者も政府も当てにせず、―― 私の老後の人生のために、私の自宅の隅々に、私のために必要な介護アシスト機器を「自分で作り、自分だけのために動かす」―― ホームセキュリティシステムの作る、と宣言しました。

なぜ私が、「自分で作る(Do It Yourself:DIY)」ことにこだわっているかいうと、私の考える「老人ホーム4.0」を本気で実現するためには、このDIY戦略がもっとも有効であると考えるからです。

まず、誰かに頼めばコストが高くなります。業者に頼んで、自宅にLANを敷設するだけでも、数十万円のオーダに乗ってしまうことがあります。ここから介助ロボットやら移動歩行器、各種のセンサーを購入すれば、さらにコストが増します。

さらに、それらのシステムを動かすための、アプリケーションプログラムを作ることも必要ですし、とどめは、そのシステムを運用(制御、保守修理)するコストも必要になります。

ざっくり、試算してみたのですが、

一一 土地付きの一軒家が、もう二軒建つ

くらいの値段になってしまいました。

　これだけのお金を払って誰かに作ってもらうくらいであれば、老人ホームで、最高クオリティの介護を受けることができるでしょう。

　しかし、コストの問題は、そんなに深刻ではないかもしれません。「少子高齢化」そのものが解決してくれるかもしれないからです。

コストは「老人ホーム4.0プラットフォーム」で解決

　1つ目はパッケージ化による低価格化です。老人ホームシステムのOSや基本アプリを詰め込んだ、「老人ホーム」プラットフォームソフトウェアWidows/Widowers*）（×Windows）が販売されるかもしれません。

*）意味は辞書で調べてみてください

　深夜の秋葉原で、高齢者の長い行列ができる……ということはならないでしょうが、オンラインソフト、または、クラウドサービスとして提供される可能性はあります。

　2つ目はコモディティ化による低価格化です。

　介助ロボット、昇降移動椅子、パワードスーツや、それにマスタ、スレーブも量産化され、コモディティ化することで、今よりもっと安価に提供される可能性もあります。例えば、イーサネットの通信カードは、30年前には24万円しましたが、今は710円で買えます（コストダウン率99.7%））。

　だからコスト面に関しては、あまり心配しなくてもいいかもしれません。

　むしろ問題は、心理（マインド）面だと思っています。

「何かあったら、誰が責任を取るんですか？」

　私たちは、「新しいモノはとりあえず否定しておく」という傾向があります。そして、その理由として「何かあった時に、誰が責任を取るのか」というフレーズで、その「否定」を正当化するのです。

　例を挙げて説明します。

【Google ストリートビューの場合】

　2007年から、Googleサイトの地図上に自宅の写真が表示されるサービスが始まり、それに対する不安感や、プライバシーの問題から、世界中で大騒ぎなりました。

日本でも、掲示板などの書き込みはもちろん、町内会、自治会の単位で、サービスの差し止めを求めて声明が相次ぎました（東京杉並区、東京町田市、大阪茨木市、高槻市その他福岡や新潟の弁護士会、国会質問など（2008年12月））。しかし、今回、判例データベースで調べたところ、実際の裁判で判決にまで至ったものはたったの1件（参考）、審理係属中のものは見つけられませんでした。

ともあれ、現時点（2015年7月）で、私は「絶対にGoogle ストリートビューを許さない」という個人や団体を見付けることはできません。あの騒ぎが、本当にウソのようです。

【ロボット掃除機の場合】

ご存じの通り、自動的に床の掃除をしてくれる円盤型の掃除ロボットです。産業ロボット大国日本にあって、この程度のロボットを作ることは、さして難しくなかったのですが、

- 仏壇にぶつかり、ろうそくが倒れ、火事になる
- 階段から落下し、下にいる人に衝突する
- よちよち歩きの赤ちゃんの歩行を邪魔し転倒させる

おそれがあり、そして、―― そのような事態になったら誰が責任を負うのか ―― が問題になって、日本のメーカが開発をためらってきたという話は、あまりにも有名です。

「『100%の安全性を確保できない』ものには手を出さない」

その精神は孤高にして気高い。

たとえ市場を失っても、その精神を貫くことは賞賛されてもいい ―― と私は思うのです。

で、たった今、検索エンジンで「国内 掃除 ロボット」で検索したら、いくつかの日本の大手家電メーカーの名前を見つけることができました。

これらのメーカーは、『仏壇とロウソクとその炎をセットで検知する画像認識機能』や、『赤ちゃんの歩行経路を推論するエンジン』を完成させ、掃除ロボットへの実装が完了したのでしょうか。ええ、私は、嫌味を言っているのです。

【自動運転車の場合】

今、騒がれているGoogleの自動走行車（Google Self-Driving Car）が登場する実に25年も前、当時、私の前の席に座っていた先輩研究員は、高速道路の自動運転アルゴリズムの開発を概ね完了していました。

実際のところ、1970年（40年前）の時点において、「日本の高速道路であれば、自動運転は実現できる」といわれていたくらいです。

しかし、この研究も、やはり、―― 交通事故が発生したら誰が責任を負うのか ――という、技術とは全く異なる観点から、消滅させられる運命にありました。

責任を負うのは誰なのか。運転手か、自動車会社か。アルゴリズムを開発したメーカー？ それとも、そのプログラムを実装した装置ベンダー？

なお、現時点において、日本の法律は、自動運転の車両の販売および使用を認めていません。

【インターネット検索エンジンの場合】

GoogleやYahoo!が登場する前、日本の企業の研究機関は、現在の検索エンジンと同程度の機能を有する検索エンジンのプロトタイプを完成していました。

ところが、こっちの方で問題となったは、―― 著作権法上の違法行為にならないか。違法行為となった場合、誰が責任を負うのか ―― という（面倒だから以下省略）。

【私の映像データ転送実験の場合】

10年前に、研究所の敷地内（私有地）で、数十台のカメラを使った大量データ（画像データ）の無線転送実験をやるためだけに、私は、法律の条文はもちろん、県や市の条例まで読み込みました。

その条文の実施事項を完全に順守するために、実験開始前にはデカデカと『通信実験中であること』を立て看板で表示し、『画像データは即日廃棄する』旨の表示をして、ようやく実験を開始することができました（繰り返しますが、私有地内の話ですよ）。

しかし、今や、街や駅やスーパーマーケットのあちらこちらに、隙間なく設置された監視カメラが、私たちの姿を記録し続けていますが、このことに血相を変えて怒っている人を見たことがありません。

この状態を考えると、―― あの時の、私のあの苦労って、一体何だったんだろう―― と、本気で泣けてきます。

エンジニアの心を殺す者たち

誤解されては困るのですが、私はこのような『サービスプロバイダや製造メーカーのマインドが間違っている』とは思っていません。

販売戦略としては決して悪くない。虎の尾は、自分でない誰か（海外のメーカーなど）に踏んでもらうのが一番です。

ただ、このような戦略が確実に壊すものが1つあります。『エンジニアの心』です。

前述のGoogleのように、『人の家の写真を取りまくった揚げ句、それをネットにアップロードする』という傍若無人の振舞いができるような、ほんの少しの『野蛮さ』があれば、と思わずにはいられません。

「まずやってみる」→「問題があれば、その時に考える」という思考形態は、私たちに欠けているものです。

「まず提案する」→「安全性はどうだ、法律はどうだ、世論はどうだ、運用はどうだ、売り上げはどうだ」→（このループが続く）→「もう、疲れたよ。やめよう」と、いう流れで、一体これまで、いくつの「若いやる気」と「野心」が潰されてきたことか。

もちろん、スピンアウト（独立してベンチャーを立ち上げる）の気概がないわれわれエンジニアの側にも責任の一端はあるかもしれません。実際、最近、ベンチャー融資の総額が増えているというニュースを見ました。これは、大変明るい話なのかもしれれません。

その一方で、失敗したベンチャーの末路が、どのような悲惨な最期を迎えているか、という話は、なかなか見つけることができません。

私が大学1年生の時、起業した父の木工会社が、連鎖倒産のあおりを食らいました。その倒産元の夫婦は、ヤクザに追い込まれて、息子たちを逃がした後、夫婦で自殺しました。あの時は、私も大学の退学を覚悟しました。

「起業家がきちんと法律通りに守られる」と信じている人は、一度、裏社会の暴力装置と、日本の自殺者の構成比率を調べることをお勧めします。

私が覚えている限り、単に「失敗すれば終わり」 ―― で済ましてくれるほど、世の中は優しくなかったですよ。

今までにない新しいコンセプト（「老人ホーム4.0」とか）の何かを始めたければ、自分のやったことに対する責任が、自分だけに戻ってくるDIYのR&D（Research and Development：研究開発）が、結局のところ、面倒が少なくて、手っ取り早いのです。

私が、自分の力でネットワークを敷設して、自分でマスタやスレーブや介護装置を設置した揚げ句、介護ロボットの誤動作で指を切り落しても、移動昇降椅子から転げ落ちて死亡しても、それは自業自得です。

それに、私が、休日にソファでビールを飲みながらプロ野球を観戦していようが、あるいは、自宅のセキュリティシステムをDIY開発していようが、それは私の自由です。他人からとやかく言われる筋合いはありません。

ただ、2つほど問題があります。技術力と資金力です。

“週末エンジニア”の存在が、技術革新の原動力に

技術力に関しましては、礼儀正しく有能な後輩の技術者たちが、プライベートにいろいろとアドバイスをくれます（無礼な上に、全く役に立たない後輩もいますが）。

資金の援助は得られませんが、ベッコフ社さんのように、面会に応じていただき、機材（EtherCATスレーブなど）を貸していただき、技術アドバイスまでいただける親切な会社あります（もっとも、何回メールをお送りしても、完全に無視されてしまった会社も、沢山ありますどねえええええええ）。

ウィークデーは企業研究員として研究開発をして、ウィークエンドには週末研究員として研究開発をするって ーー 江端ってアホじゃないのか？ ーーと思われる方も多いかもしれません。

しかし、私の回りには、こういうエンジニアが結構沢山います。

そして、その人達の貢献が、現実に、現在の世界の技術を進歩させる、大きな推進力になっていたりするんです。

このお話は、また別の機会にさせていただきたいと思います。

⇒本連載のバックナンバーはこちら

Profile

江端智一（えばた ともいち）

日本の大手総合電機メーカーの主任研究員。1991年に入社。「サンマとサバ」を2種類のセンサーだけで判別するという電子レンジの食品自動判別アルゴリズムの発明を皮切りに、エンジン制御からネットワーク監視、無線ネットワーク、屋内GPS、鉄道システムまで幅広い分野の研究開発に携わる。

意外な視点から繰り出される特許発明には定評が高く、特許権に関して強いこだわりを持つ。特に熾烈（しれつ）を極めた海外特許庁との戦いにおいて、審査官を交代させるまで戦い抜いて特許査定を奪取した話は、今なお伝説として「本人」が語り継いでいる。共同研究のために赴任した米国での2年間の生活では、会話の1割の単語だけを拾って残りの9割を推測し、相手の言っている内容を理解しないで会話を強行するという希少な能力を獲得し、凱旋帰国。

私生活においては、辛辣（しんらつ）な切り口で語られるエッセイをWebサイト「こぼれネット」で発表し続け、カルト的なファンから圧倒的な支持を得ている。また週末には、LANを敷設するために自宅の庭に穴を掘り、侵入検知センサーを設置し、24時間体制のホームセキュリティシステムを構築することを趣味としている。このシステムは現在も拡張を続けており、その完成形態は「本人」も知らない。

本連載の内容は、個人の意見および見解であり、所属する組織を代表したものではありません。

関連記事

エンジニアは好きなことだけやってる？ そんなのウソです
エレクトロニクスエンジニアの雇用基盤が揺らいでいる。国内企業は人員削減や賃金カット、工場閉鎖、事業売却などのリストラを加速。たとえ大企業でも、以前のように安定した雇用を期待しにくい時代だ。竹内氏は、大企業の人事制度の限界を指摘しつつも、「エンジニアがスキルアップする場としてはメリットがある」と語る。

“できるエンジニア”は、こう行動する〜今日から始めたい10のルール〜
“できるエンジニア”と言われる人たちの行動には、昔も今も、ある共通点がありました。今回は、その10個の共通点をご紹介します。

若きエンジニアへのエール〜入社後5年間を生き残る、戦略としての「誠実」〜
「新卒から入社5年目ぐらいまでの新人、若手」という方は、たぶん社会人として最も辛く、厳しい時期にいる方だと思います。社会の矛盾、不合理、理不尽、不条理という名の弾丸が、複数の機関銃から一斉掃射される、悲惨な戦場にいるかのような、恐ろしく辛い時代です。その間の無防備な時代を凌ぐには、「戦略」が必要です。私は、新人、若手の皆さんに、「誠実」、「温厚」、「寄生」という3つの戦略を授けたいと思います。

エンジニア史に残る10人の女性たち（前編）
工学の分野では、古くから女性が活躍していた。プログラミング言語「COBOL」の誕生、世界初のプログラミングコード、アポロ11号の月面着陸……。サイエンスやテクノロジーの歴史に残る出来事の舞台裏で活躍した女性エンジニアたちに焦点を当てる。