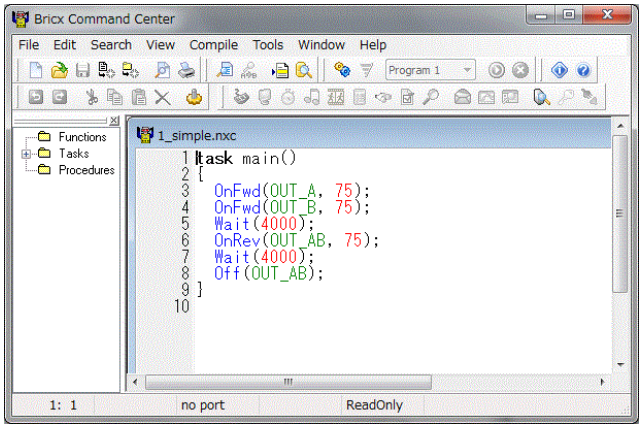
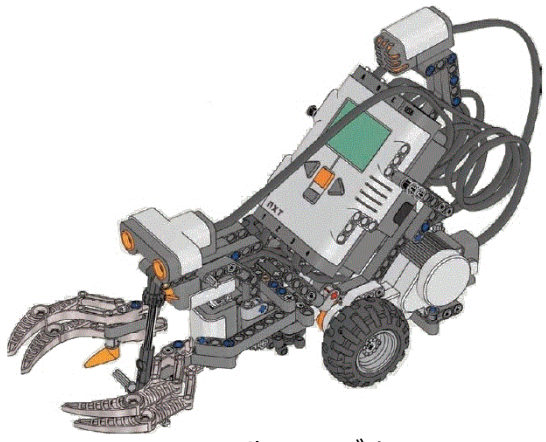
LEGO Mindstorms NXT は、簡易なソフトウェアから高度なものまで、ロボット工学による幅広い組込み・制御の学習・教育が可能なキットです。

2012年5月1日発刊



BricxCC画面



Tribot モデル

BricxCCおよびNXCによるチュートリアル（教育カリキュラム）

Produce By Takayori Takamoto　Ph.D.

BricxCCとNXCによるソフトウェア開発環境を易しく紹介するドキュメントです。

TABrain

2011/09/09版

もくじ

※NXC と 標準C言語 との違いについては、別途「NXCプログラマーズガイド」をご覧ください。

Ⅱ．BricxCC関連情報

1．BricxCCのインストール

BricxCC紹介Webサイト
BricxCCインストール手順
BricxCC（NXC)のヘルプWebサイト

2．BricxCC操作ガイド

立上げ操作
BricxCC画面メニュー構成説明
プログラムのコンパイル・実行
メニューバー
ツールバー
マルチ・エディタ画面の設定
立上げ省略設定
コンパイル初期設定
ツールバー関連の表示オプション
プログラム・エディタ
プログラム・エディタの画面オプション
エディタ・ページの設定
マクロ機能
表示カラー設定
エディタ・オプション設定
キー割り当て設定
コードテンプレート設定
ファイル拡張子認識
ファイル・メニューオプション
エディタ・メニューオプション
検索メニューとツールバー
ブックマーク機能
エディタ編集補助機能
ドラッグ・ドロップ・プログラミング
コード・エクスプローラ機能
マクロ管理
コンパイル・エラー表示機能
ヘルプ機能

Ⅲ.　NXCプログラミング

1. はじめてのプログラム作成

ロボットの組み立て
「Bricx Command Center」の起動
プログラムの作成
プログラムの実行
プログラム内のエラー（誤り）
スピードの変更

2. より興味のあるプログラム作成

旋回プログラム
コマンドの繰り返し
コメントの追加

3. 変数の使い方

渦まき（螺旋）起動
乱数（ランダム数）

4. 制御構造

if ステートメント
do ステートメント

5. センサーの使い方（基礎）

センサーの待機
タッチ・センサーの反応
ライト・センサー（光センサー）
サウンド・センサー（音センサー）
超音波センサー

6. タスクとサブルーチン

タスク
サブルーチン
マクロの定義

7. ミュージックの作成

サウンド・ファイルの再生
ミュージックの再生

8. モータの詳細設定

減速停止
上級コマンド
PID 制御

９. センサーの使い方（応用）

センサーのモードとタイプ

回転センサー

ひとつの入力ポートに複数センサー接続

１０. 並列タスク

間違ったプログラム

危険領域と mutex変数

セマフォーの利用

１１. ロボット間の通信

マスターとスレイブ間のメッセージ送信

送信数の承認（ack）

直接的なコマンド

１２. その他のコマンド

タイマーの使い方

ドットマトリックス・ディスプレイ
（NXT画面表示）

ファイルシステム

Ⅳ.　カリキュラム編

1．ロボット教育カリキュラム

① ロボット教育カリキュラムとは

② ロボット教育カリキュラム事例

2．ロボット教育カリキュラム思考

① NXC（Not eXactly C）とは

② レゴ・マインドストーム教育機関向け教材

③ LEGO Mindstorms NXTの開発環境

④ LEGO Mindstorms NXTの関連サイト

⑤ LEGO Mindstorms NXTの拡張

⑥ NXT プログラミング教本概要

⑦ マインドストーム学習カリキュラム例

【参考資料】

参考資料① カリキュラム導入・運用シナリオ

参考資料② LegoMindstorms NXT関連本

参考資料③ ロボット教育で学ぶ情報技術

はじめに

- レゴ・マインドストームは多くの教育現場で利用されるようになり、教育機関だけでなく、技術会社の教育用としても導入されるようになり、その勢いも年々増加している。
- 既に多くの出版物やWebサイトでの情報も豊富で、さまざまなレゴ・マインドストームに関する技術スキルアップやロボット関連の一般的な教育スキルアップも得られるようになっている。
- また、レゴ・マインドストームを使った競技会も盛んに開催されるようになり、ローカルでは多くあり、国際的にも３つほどの大きな競技会が開催されている。
- ここでは、レゴ・マインドストームを動かすプログラム言語のひとつであるNXC（Not eXactly C）と呼ばれるC言語ライクなものを使った学習カリキュラムをまとめている。
- NXCは、オープン言語で、無料で開発統合環境であるBricxCC上で簡単にマインドストームのプログラムを開発し、組込み、いろいろと楽しむことができる。
- このBricxCCとNXCは、John Hnasen氏らによって開発されて、フリーソフトとして提供されている。
- 本テキストでは、ロボット教育、レゴマインドストーム関連情報、それにBricxCCのインストールと操作ガイド、NXCのチュートリアルガイド、それに教育カリキュラムと幅広くまとめている。
- ロボットを学ぶ初心者から中級者までを対象として、分かり易く図や補足説明を加えて、まとめている。

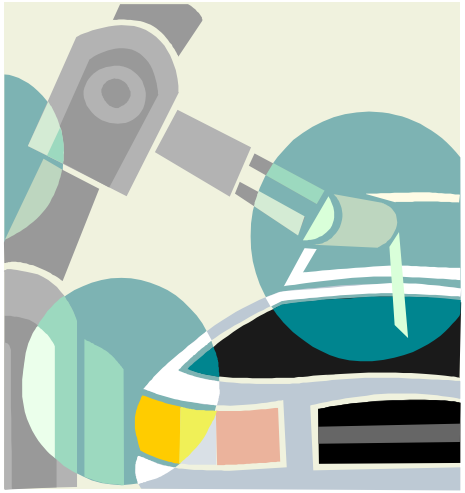
2011年 8月　記:高本孝頼

Ⅰ．　ロボット教育とNXT

ロボット教育と
NXT情報および開発環境ほか

１．ロボット教育関連

① ロボットとは
② ロボット教育とは
③ ロボット教育で学ぶもの
④ ロボット教育について
⑤ ロボット教育の場

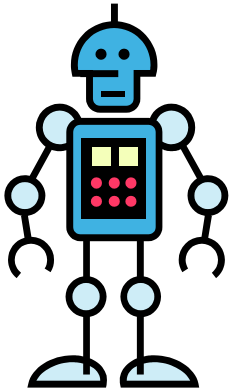


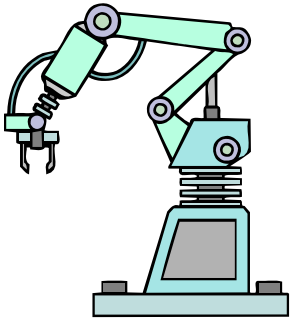
① ロボットとは

高本作成（2011.05）

ロボットを学ぶ上での広範囲な知識は幅広く、何を知るべきか、何を教えるべきかを明確にした上で、
学んでもらうことが必要では・・・・

ロボットの活躍の場は、ひろがりを見せている。
- 家庭内ロボット（監視など）
- ホビーロボット・癒し系ロボット
- 医療・介護ロボット
- 産業用ロボット
- 調理ロボット
- ロボットカー
- 軍事用ロボット
- エンターテイメント用ロボット
- 教育用ロボット





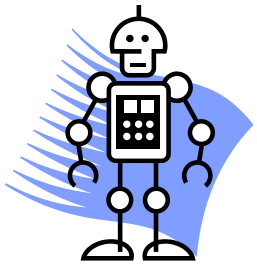
ロボットの活躍の場は、特に３K（きつい、汚い、危険）にある

実用の日本製産業用ロボットは世界一
研究から実用への展開をしつつある段階が多い

※日本でのロボットの活躍の場は、法規制などの課題があり、小さい。
　ヨーロッパでの規制が少ないために、研究者が移住するなど。

④ ロボット教育とは

- ロボットとは、計測・制御の機能を使った人間や動物の振る舞いに似せた動きをさせたもの
  - ロボット工学によって学ぶべきこと
    ア　計測・制御の機能
    イ　各種制御システム（組込みソフトウェア）
    ウ　操作と安全管理など



- **制御（Control）：**
  - 機械などが目的にそって動くように操作・調整すること（広辞苑）
    - シーケンス制御、フィードバック制御、コンピュータ制御（指導要綱記述）
    - 家電製品（計測）・自動車（速度制御）・航空機（自動操縦）などに利用

- **組込み（Embedded）：**
  - 特定の機能を実現するために機械や機器に組み込まれるコンピュータシステム（コンピュータ用語辞典）
    - マイコンの組み込み技術（指導要綱記述）
    - 常にCPUの高機能化・コンパクト化・高密度化によって、常に技術者のスキルアップが必要

③ ロボット教育で学ぶもの

- ロボット工学
  - ロボット制御の基礎であるフィードバック制御などを学習。さらにコンピュータ制御も併せて学習可能。
  - ロボット工学では、機械工学・電子工学・情報工学など、総合的に組み合わされた技術の習得が可能。
  - モジュール化（ブラックボックス化）されたロボットを活用することで、部分的な勉強や学習が可能

- 制御・組込みソフトウェア
  - コンピュータ制御による各種センサや動力部を使ったソフトウェア開発技術習得
  - 同期をとった並列処理や群ロボット開発の習得、無線通信処理の習得
  - 現場の状況に応じたシミュレーション対応によるソフト開発習得

④ ロボット教育の場

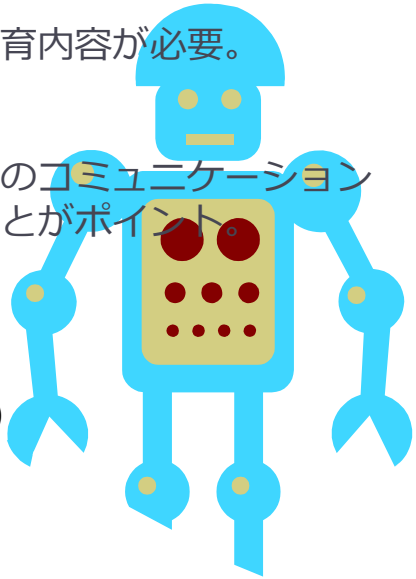
- 教育の場の提供
  - 授業だけでなく、課外活動や地域活動、さらに競技会などの参加
  - さらに、資格取得や有用な各種検定を促進
- 期待する教育効果
  - 技術スキルアップ（コンピュータ制御や組込みソフトウェアの技術）
  - 単に知識習得だけでなく、コミュニケーション能力を高め、協調性、創造性なども一緒に収得
- 最新技術との触れ合い
  - 日進月歩が早いモノづくりの最新技術の習得
  - 日本国内だけでなく国際的な視点でのモノづくりの視点で対応

⑤ ロボット教育について

- 教育の目的・主旨について
  - ロボットによる最新技術との触れ合い。センサ技術やアクチュエータを含むコンピュータ制御やソフトウェア組込み技術の習得。グループ作業などを通じての創造力・コミュニケーション能力・協調性を身に付ける教育の場の提供。
- 時間的な配分について
  - 工業高校だと3~5時間程度から、10時間を超える授業まで、さまざまな展開が行われている。
- 専門的な配分について
  - ロボット教育において、易しく専門用語を教え、ハードルを低くした教育内容が必要。
- 教育環境について
  - 理論だけでなく、実践して学ぶことが重要。さらにグループ（団体）でのコミュニケーション能力やプレゼンテーション能力を高めるための競技なども取り入れることがポイント。
- 教材に使うロボットについて
  - 汎用的で、廉価、かつ、応用・拡張ができるロボット教材が最適。

（汎用的とは、機械や電子・電気、および情報でも使えるロボットが最適）

  - 教材として、教える立場の資料が揃っていることが重要。



2．マインドストーム関連情報

① レゴ・マインドストームとは
② レゴ・マインドストーム教育機関向け教材
③ LEGO Mindstorms NXTの開発環境
④ LEGO Mindstorms NXTの関連サイト
⑤ LEGO Mindstorms NXTの拡張
⑥ NXT プログラミング教本概要

① レゴ・マインドストームとは

- LEGO Mindstormsは、米国昌チューセッツ工科大学（MIT）によって開発されたロボット工学を効果的に学習することができるキットです。

- 1998年にRCXが販売開始され、2006年にNXTが、さらに2009年にNXT2.0が販売されています。

- 世界的に教育現場で利用され、多くの競技会・コンテストなどが開催されています。
- Web上では、Mindstormsに関する多くの情報や、開発環境の提供などが豊富に掲載されています。

② レゴ・マインドストーム教育機関向け教材

【レゴ 社マインドストームとは】

米国・マサチューセッツ工科大学（MIT）の研究成果をベースに開発されたLEGO社のロボット教材で、1999年に初期バージョンのRCXが販売開始、2006年にNXTが登場、さらに2009年にNXT2.0が最新版として販売されている。現在、日本では教育版が日本語化されて販売され、玩具版は英語版のみで販売されている。

【レゴ社マインドストームの教育教材とは】

1）多くの学校教育教材として実績があり、小学校から、中学校、高校、大学、さらには企業においても幅広く利用
2）自由に組み立て可能なブロックで、作っては崩し、考えては組み立てすることで、想像力・創造力を刺激
3）プログラミングによるロボット制御を学習し、アイデアを創出。自分なりのプログラムを動かすことが可能
4）モノづくりの実体験が可能で、 機械工学、機構工学、電子・電気工学、情報工学まで含めた統合的な学習が可能
5） 最新のセンサ技術やモータ（アクチュエータ）技術を利用し、組込み・制御のプログラム開発を学習可能

【レゴ社マインドストームの教育教材導入メリット】

1）自由な発想で、固定概念にこだわることなく、観察力・想像力・独創力・思考力を育成できる
2）課外活動や競技会活動を通じて、コミュニケーション能力やプレゼンテーション能力を育成できる
3）ブロックの組み立てから、センサやアクチュエータを稼働することで、総合的な学習ができる
4）モノづくりの基本的な学習が学べる

③ LEGO Mindstorms NXTの開発環境

高本作成（2011.05)

開発環境	概要説明	参考本
①ROBOLAB ②NXT-SW ③NXT-G	・LEGO社提供による**ビジュアル系プログラミング開発環境** ・ROBOLABは、LabVIEWをベースとするビジュアル系（グラフィック系）開発環境。詳細な設定まで対応可能。 ・NXT-Gは、英語のみの玩具版で提供されている開発環境 ・NXT-SWは、日本語版もある教育版で提供されている開発環境	・Programing LEGO MINDSTORMS NXT ・LEGO MINDSTORMS NXT-G Programming Guide ・入門LEGO MINDSTORMS NXT 第2版
④NXC（BricxCC）	・オープンソフトウェアとして**C言語に近い開発言語** ・BricxCCは、NXCほか多くの開発言語でNXTを稼働するプログラム統合環境	・LEGO MINDSTORMS NXT POWER PROGRAMMING ・実践ロボットプログラミング（近代科学社）
⑤Microsoft Robotics Studio（MSRS）	・マイクロソフト社が提供するロボット開発のための開発環境 ・NXTをはじめ、多くの既存ロボットの開発環境を提供 ・ビジュアルプログラム言語やC#言語などの開発言語をもち、バーチャルなシミュレーション環境を持つ	・Microsoft Robotics Studio プログラミング ・Robotics Developer Studio入門（工学社） ・Programming Microsoft Robotics Studio
⑥MATLAB & Simulink ⑦LabVIEW	・数値解析シミュレーションソフトのMATLAB & SimulinkやLabVIEWなどによるビジュアル系での高度なロボット制御を実現	・LabVIEWで学ぶ [最新] LEGO Mindstorm s NXT 入門

備考
※ LEGO MINDSTORMS NXTの開発環境は豊富に提供されていて、オープンソースなど、Web上でも多くの事例サンプルが紹介されている。

④ LEGO Mindstorms NXTの関連サイト

高本作成（2011.05）

項目	内容	Webサイト	概要説明
1	LEGO社 サイト	英語版　mindsotrms.lego.com 日本語版　www.legoeducation.jp/mindstorms mindstorms.lego.com/Overview/NXTreame.aspx	レゴ・マインドストーム公式サイト（英語版） レゴ・マインドストーム公式サイト（日本語版） NXT仕様や開発ツール紹介サイト
2	Webサービス 関連	LUGNET　www.lugnet.com Brickshelf　www.brickshelf.com Peeron　www.peeron.com	レゴブロック国際的ユーザグループサイト ユーザ画像投稿サービスサイト パーツやセット紹介サービスサイト
3	プログラム 関連	www.ni.com/academic/mindstorms msdn.microsoft.com/robotics/ bricxcc.sourceforge.net/nbc bricxcc.sourceforge.net/nqc www.robotc.nxt www.mindsgualls.net lejos.sourceforge.net lejos-osek.sourceforge.net	ナショナルインストゥルメンツ社LabVIEWサイト マイクロソフト社 ロボット BricxCC（レゴ社ロボットの統合開発環境）NXC BricxCC（レゴ社ロボットの統合開発環境）NQC カーネギーメロン大学発C言語風ロボット開発環境 .NET用のNXTプログラムライブラリ。C#開発 Java開発環境。オリジナルファームウェア使用 C/C++開発環境。日本語サイトあり。NXT対応済
4	設計図・CAD CG関連	ldd.lego.com www.ldraw.org www.lm-softwaare.com/mlcad/ www5a.biglobe.ne.jp/~mutchy/lego/mlcad/ bricksmith.souceforge.net philohome.com/mxtidraw/nxttidraw.htm	レゴ社によるバーチャルレゴブロックツールツール レゴブロック用CADソフト MLCad：LDrawベースのグラフィカルなCADソフト 同上日本語サイト MLCad：LdrawベースのグラフィカルなCADソフト NXTに対応したLdraw用追加パーツセット

参考
※ 入門LEGO MINDSTORMS NXT【第2版】大庭信一郎著（ソフトバンククリエイティブ㈱）

⑤ LEGO Mindstorms NXTの拡張

NXTの接続コネクタを理解することで、市販のセンサ類やモータ類を接続し、NXTを拡張することが可能になる。

NXTコネクターの意味

ピン番号	色	入力（センサ関連）		出力（アクチュエータ関連）	
		名前	説　明	名前	説　明
1	白	AN	アナログ入力	M1	NXTバッテリー
2	黒	GND	グラウンド	M2	NXTバッテリー
3	赤	GND	グラウンド	GND	グラウンド
4	緑	4.3V電力	4.3V	4.3V電力	
5	黄	DIGI0	デジタル入力	TACHO0	
6	青	DIGI1	デジタル入力	TACHO1	

高本作成（2011.05）

項目	内容	拡　張　概　要	拡　張　利　用
1	センサ関連	接続センサ（ 0 または　1 ） 抵抗センサ（ 5V： 0 ～ 1023 ）<アナログ・デジタル変換> 電位差センサ 電圧センサ 4.3V 電力センサ	スイッチ・タッチセンサ 温度センサ・光センサなど アナログセンサ接続 バッテリ測定 ホールセンサ
2	アクチュエータ		
3	その他接続	I2C バス コミュニケーションコミュニケーション	カメラ接続など

参考
※Extreme NXT【 Extending the LEGO MINDSTORMS NXT to the Next Level

⑥ NXTプログラミング教本概要

LEGO Mindstorms NXTを動かすには、ビジュアル系か言語系による開発環境でプログラミングして実行させる必要がある。ここでは、この２つの開発環境を教える本を参考に、何を教えていくか、またその違いでの教育の概説をまとめている。

高本作成（2011.05）

項目	分類	NXT-G（ビジュアル系）※１	NXC（言語系）※２
1	前提	プログラムとは、ロボットの構築	プログラミングとは、ＮＸＴとＰＣの接続方法
2	基礎	プログラム作成、簡単な事例紹介、ブロック、ワイヤー	プログラミング文法、アルゴリズム、設計図 プログラム作成・アップロード（転送）・実行
3	データ	数値、ロジック、テキスト、および入力と出力	
4	プログラム基本	ループ（繰り返し）、判断・分岐・スイッチ、待ち時間	Ｃ言語文法（データ型、演算子、選択構造、反復構造、配列、関数、＃define文、外部変数とスコープ）
5	センサ	タッチセンサ、光センサ、カラーセンサ、音センサ、超音波センサ、ジャイロセンサ、加速度センサ、地磁気センサほか	同左
6	モータ	モータ制御（前進・後退・回転ほか）	同左
7	応用	数式処理、キャリブレーションほか	シングルタスクと並列タスク
8	拡張展開	記録動作、移動パワー制御、	ライントレース、競技ロボット、複数ロボット制御
概　説		初心者向きで、プログラム言語を覚えることなく容易にロボットの制御が学べる。 ただし、ビジュアル系プログラム開発環境は、将来的に学んだことが応用できることは少ないと考える。複雑なロボット制御には向かない。	中級者・上級者向きプログラム開発言語。統合開発環境（BricxCCなど）と一緒に利用すればより簡単にプログラム開発ができる。 細かなプログラム制御などが学べ、将来的にもプログラム開発言語の利用価値は出てくる。

参考
※１：Programming LEGO MINDSTORMS NXTほか
※２：実践ロボットプログラミングほか

Ⅱ．BricxCC関連情報

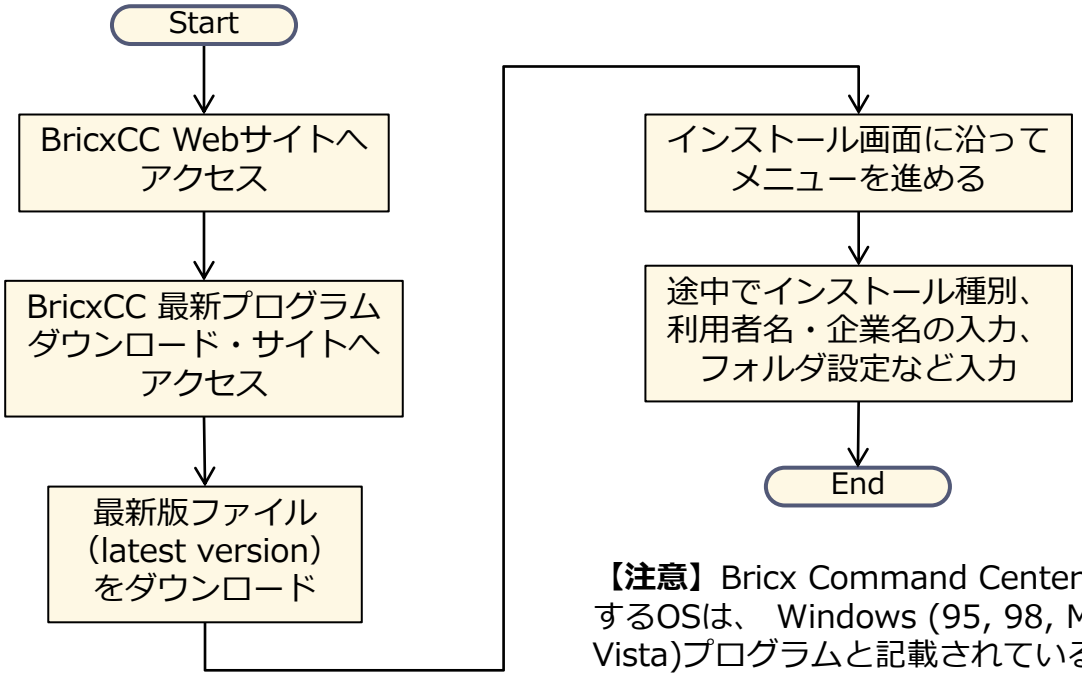
BricxCCのインストールと操作ガイド

1．BricxCCインストール

BricxCCは、レゴ・マインドストームを操作するプログラム統合開発環境としては、長年に渡って提供されてきていて、世界的に愛好者に利用され、フリーソフトとしても、ユーザ層が多いものとなっている。

ここで、BricxCCを教育用に選択した理由は、個人的な学習でも利用しやすく、かつ簡易なサンプルを使ったものから、センサーの高度なものまで利用でき、さらに音楽や画像処理、ライトセンサーなどのLED処理など、幅広くプログラムの組込み・制御が学習・教育できる点を重要視した。

ここでは、BricxCCのWebサイトからのインストール方法と、関連する情報をまとめている。

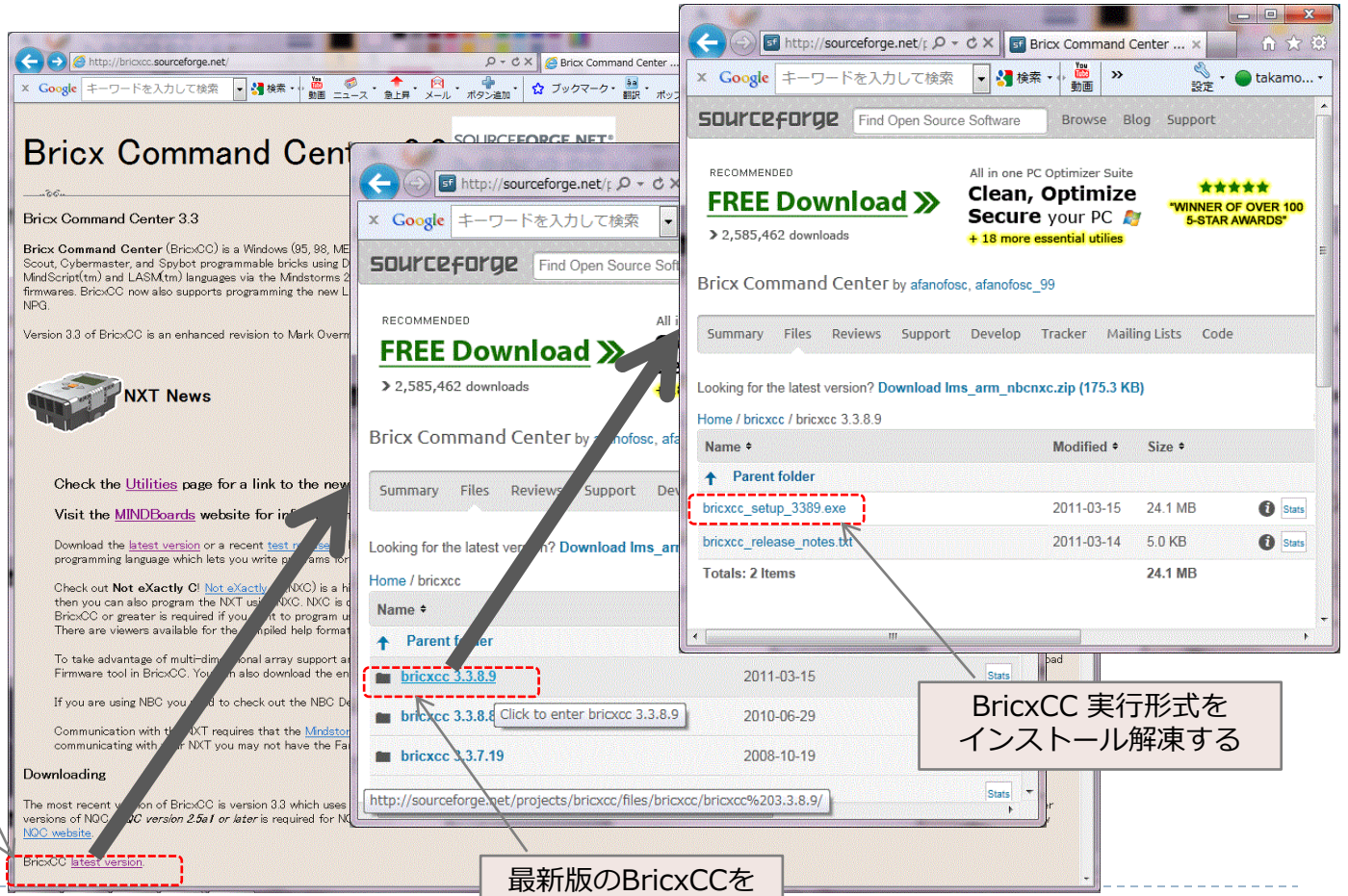


【注意】Bricx Command Center (BricxCC) の稼働するOSは、 Windows (95, 98, ME, NT, W2K, XP, Vista)プログラムと記載されているが、著者はWindows７で稼動させていて特に問題は生じていない。

BricxCC紹介Webサイト

BricxCC(Bricx Command Center)のダウンロードサイトは、
こちら　**http://bricxcc.sourceforge.net/**　から、つぎのようにたどることができるが、
直接のダウンロードファイルの一覧表となっている次のサイト
　http://sourceforge.net/projects/bricxcc/files/bricxcc/ から、最新版を選択してインストールすることもできる。

【注意】インストール解凍時点で表示される「実行」もしくは「保存」のいずれでもインストールが行うことができる。
「保存」を選択した場合には、一旦圧縮ファイルをカレント（現在）フォルダに保存することになり、その後「ファイル実行」が必要となる。

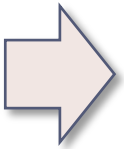


T.Takamoto 2011.08.25

TABrain

BricxCCインストール手順（１）

Windows7でダウンロードした場合、以下のような画面にて、ダウンロード状態が表示される。



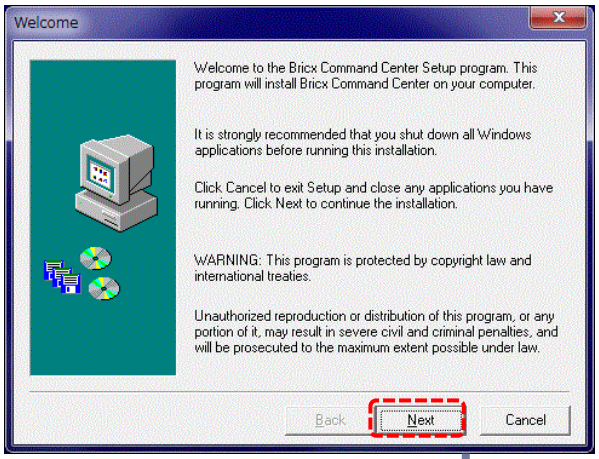
【注意】現時点（H23.08.01)での最新版は、BricxCC で、このダウンロード版（exeファイル）の容量は23Mバイトある。

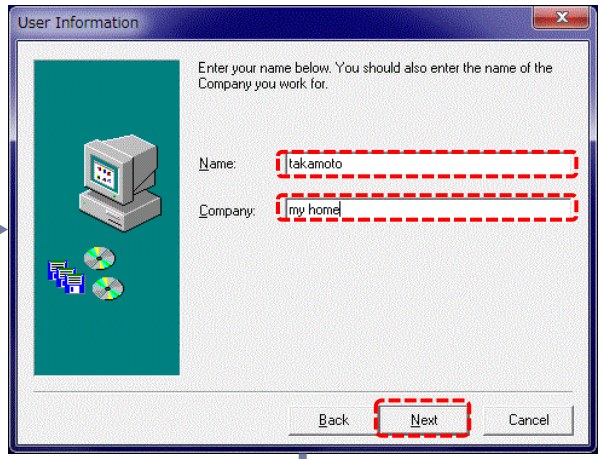
「実行」ボタンをヒットすると、「Bricx Command Center 3.3」のセットアップ・プログラムを実行し、インストールが開始される。（次ページ）
この場合、実行前に、他のプログラムを終了させておくことを推奨する。

【注意】ただし、インストールする際に、ハードウェア（CPU仕様やハードディスク容量）やソフトウェア（OS）などを確認する必要がある。

BricxCCインストール手順（２）

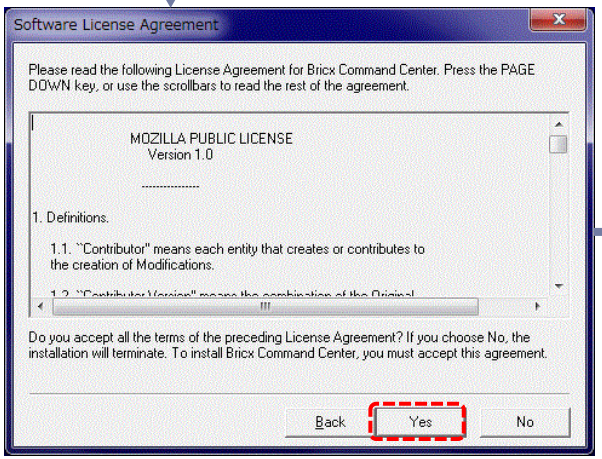
BricxCCのインストールは、特に問題なく、以下の手順で簡単にインストールができる。



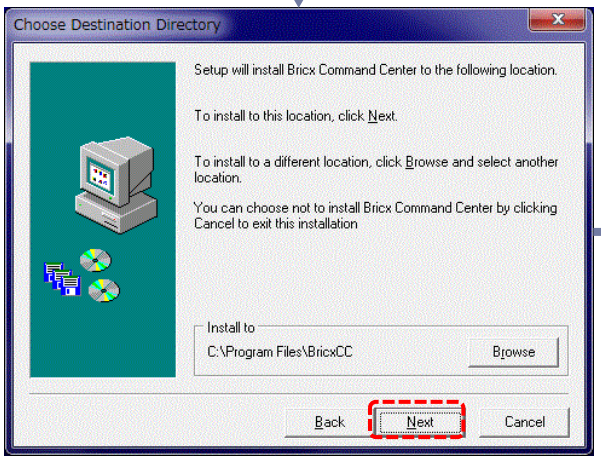


利用者の名前と企業名を入力する。

名前と会社名を記入



ライセンスの承諾となる。「Yes」を押して、次に進む。



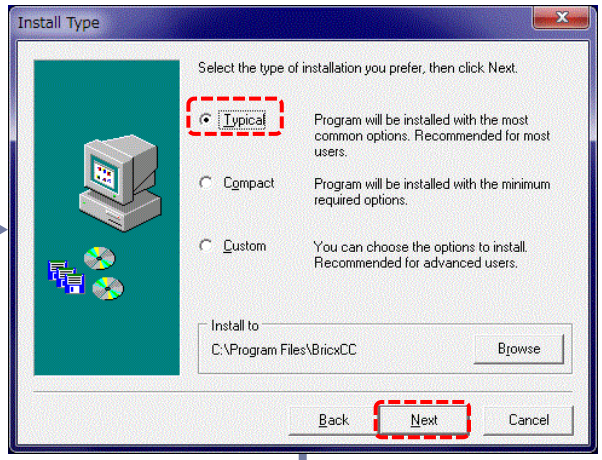
次頁

インストール先フォルダを設定

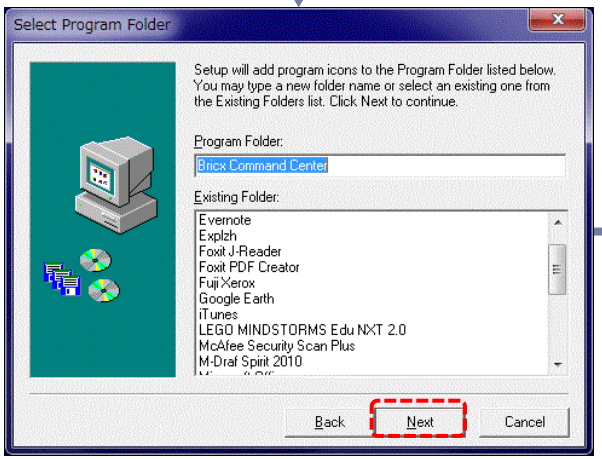
BricxCCインストール手順（３）

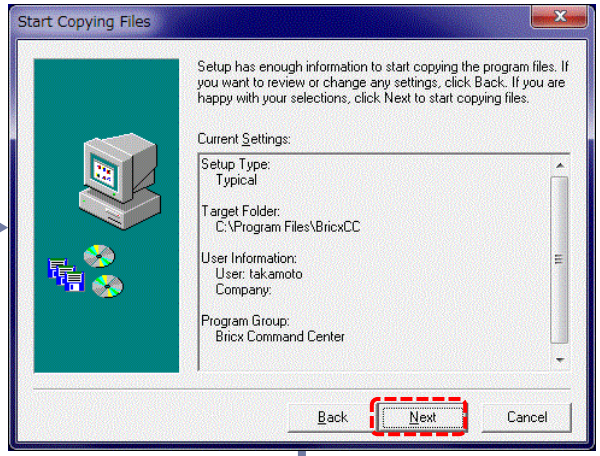
「Compact」は、最少サイズのインストールとなる。
「Custom」は、オプションを選択してインストールする。

前頁

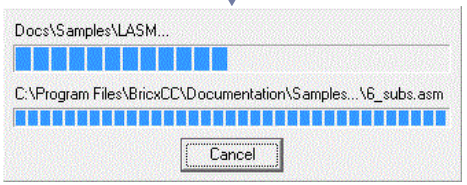


初心者においては
「Typical」で構わない。





この状態でインストール
が実施される。

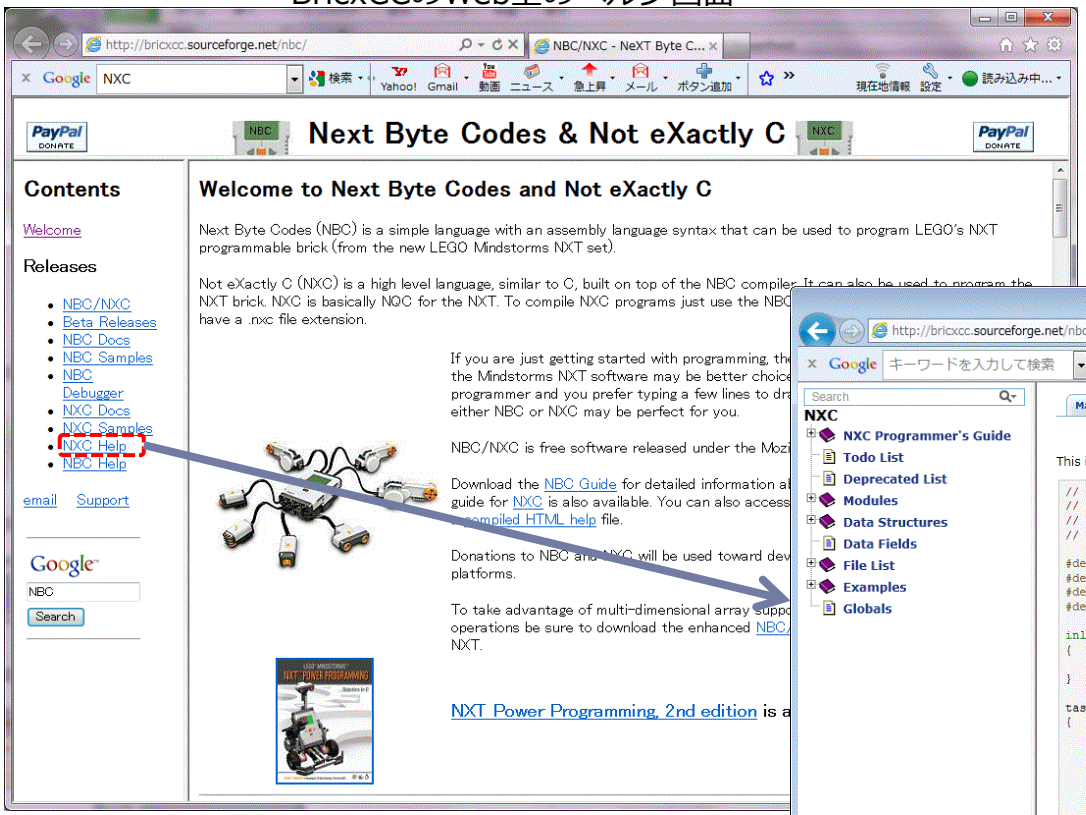


Bric x CC（NXC）のヘルプWebサイト

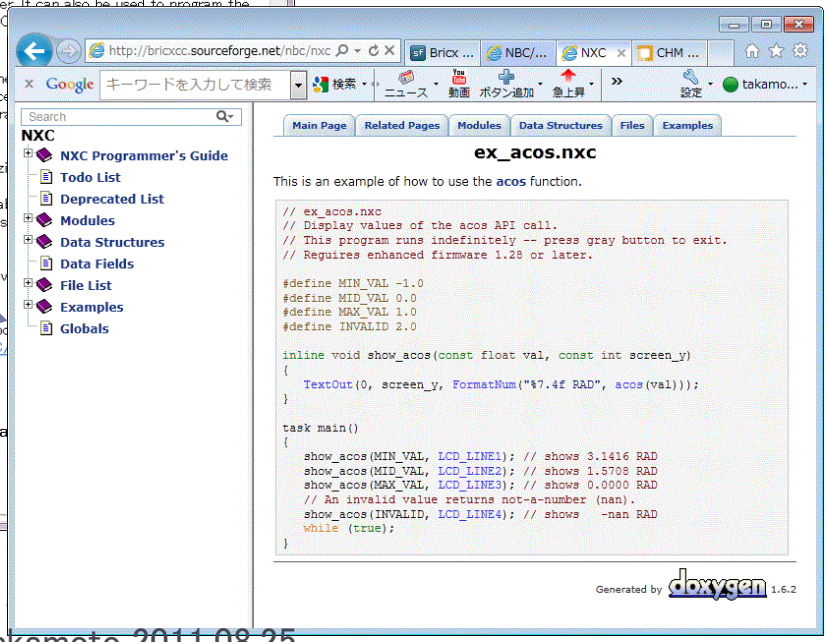
以下の画面サイトは、BricxCCにおけるWeb上のヘルプサイトで、NXCなどの文法から関数群説明などが詳細に掲載され、さらにいろいろと関連サイトにリンクされ、サンプル・プログラムなどが豊富に掲載されている。

http://bricxcc.sourceforge.net/nbc/

BricxCCのWeb上のヘルプ画面



NXCプログラミングヘルプ画面



T.Takamoto 2011.08.25

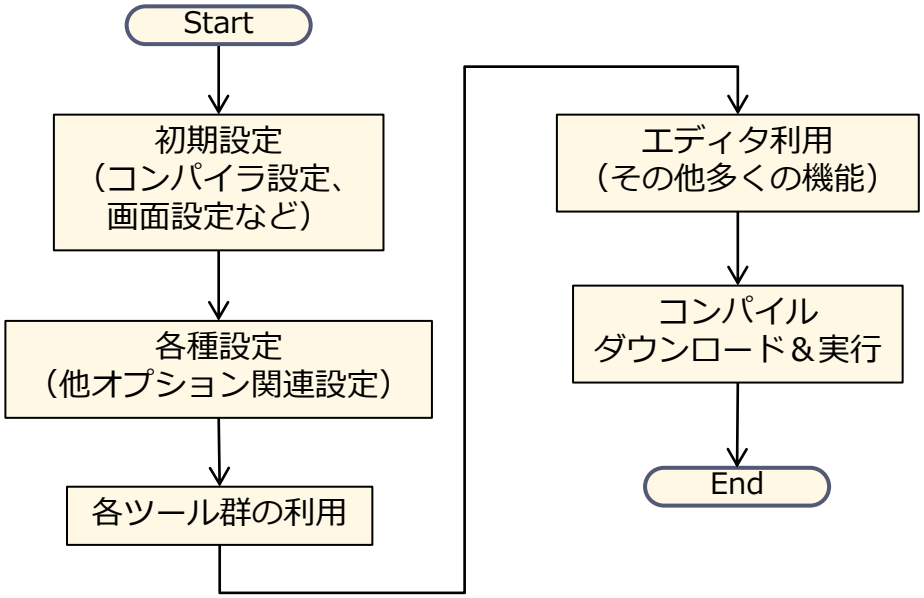
TABrain

2．BricxCC操作ガイド

ここでは、BricxCCの初期設定や立上げ画面の紹介、およびメニューに表示されている多くの機能やツール群、それにオプション関連の説明を行っている。

環境設定を知ることで、より使い易い環境でプログラム開発ができるようになる。

一度は、資料を眺めて、操作してみてほしい。



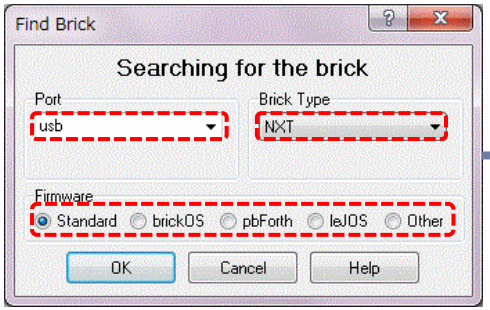
立上げ操作

BricxCCの立上げは、右図のようなBricxCC起動アイコンをクリックすることでプログラムが起動する。

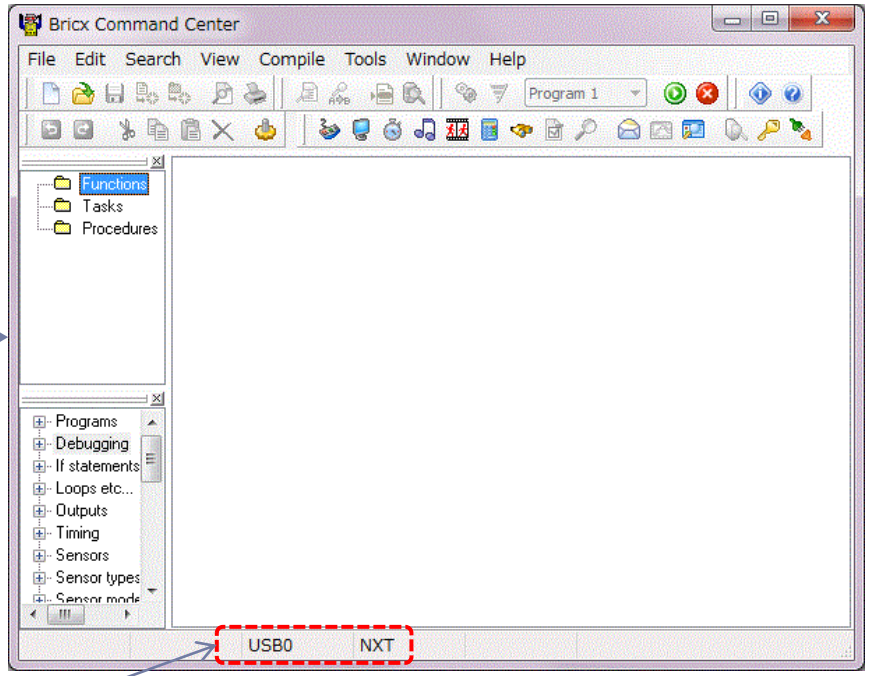
BricxCCを立上げた場合、以下のような「Find Brick」ウィンドウが立ち上がる。
この場合、**Port**はNXTへの接続ポートを選択する。最初は、USBポートで行うか、Bluetooth設定での接続を選択する。
Brick Typeは、NXTを選択する。さらに、**Firmware**では、初期設定での「Standard」を選択して、「OK」ボタンをヒットしてBircxCCを起動する。

BricxCC起動アイコン（メニュー）

Find Brick画面



BricxCC画面

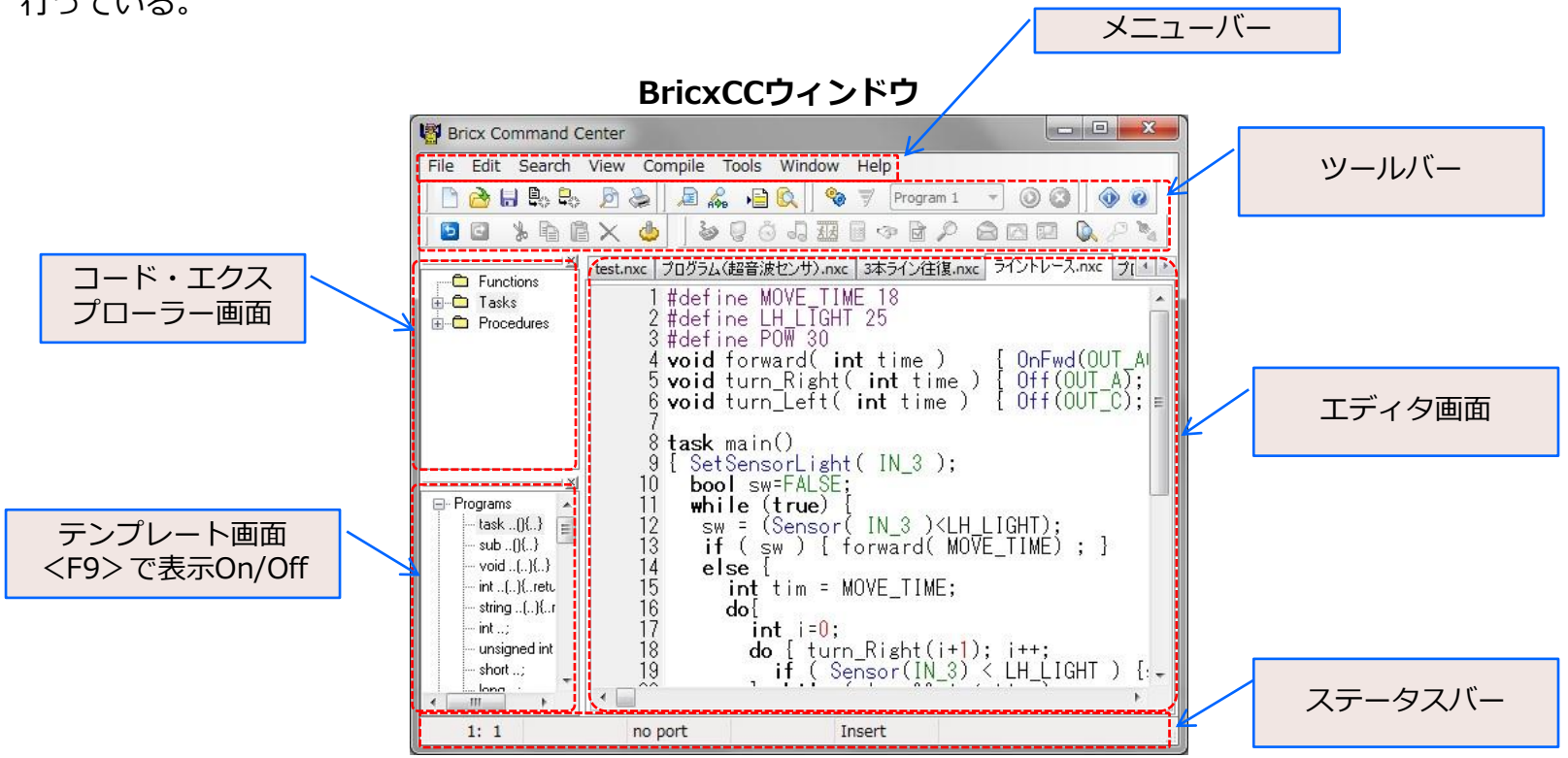


【注意】必ずしもNXTが接続されていなくてもBricxCCの立上げは可能。

【注意】NXTが接続ができた場合には、接続ポートとNXTが表示される。

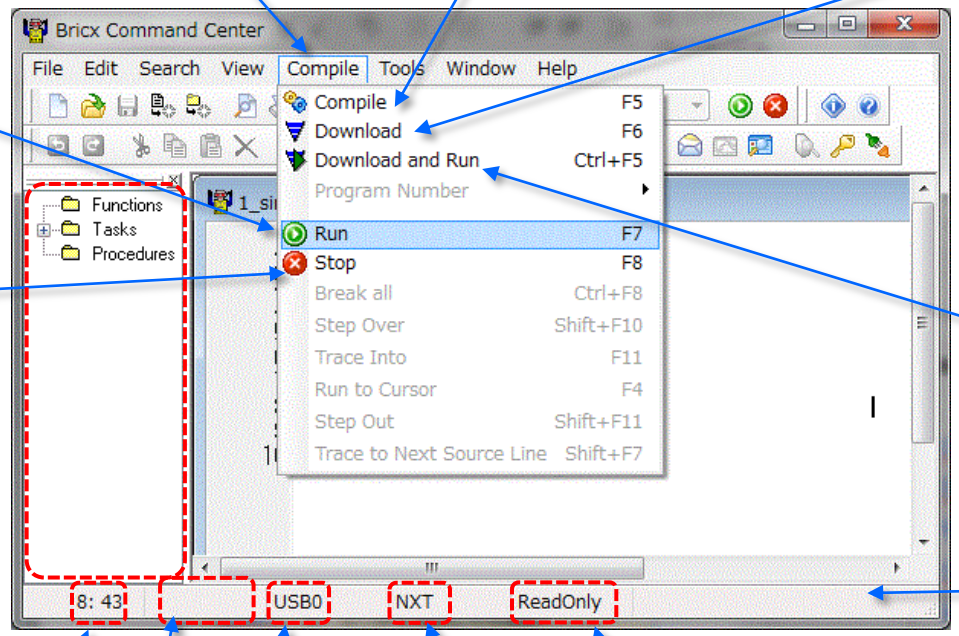
BricxCC画面メニュー構成説明

BricxCC画面に表示されるメニューおよび各種画面の説明を
行っている。



プログラムのコンパイル・実行

- BricxCC画面上の「Compiler」から「Run」を実行、もしくはNXT画面上で実行する。



メニューバーの「Compile」をヒット

「Compile」：編集中のプログラムをコンパイル

「Download」：コンパイルした実行プログラムをNXTへダウンロード（組込み）する

「Run」：NXT上のプログラムの実行

「Stop」：j実行中のNXT上のプログラムを停止

「Download and Run」：コンパイルした実行プログラムをNXTへダウンロード（組込み）し、実行まで行う

ステータスバー

カーソルの位置（行：桁）を表示

NXTとの接続状態を表示

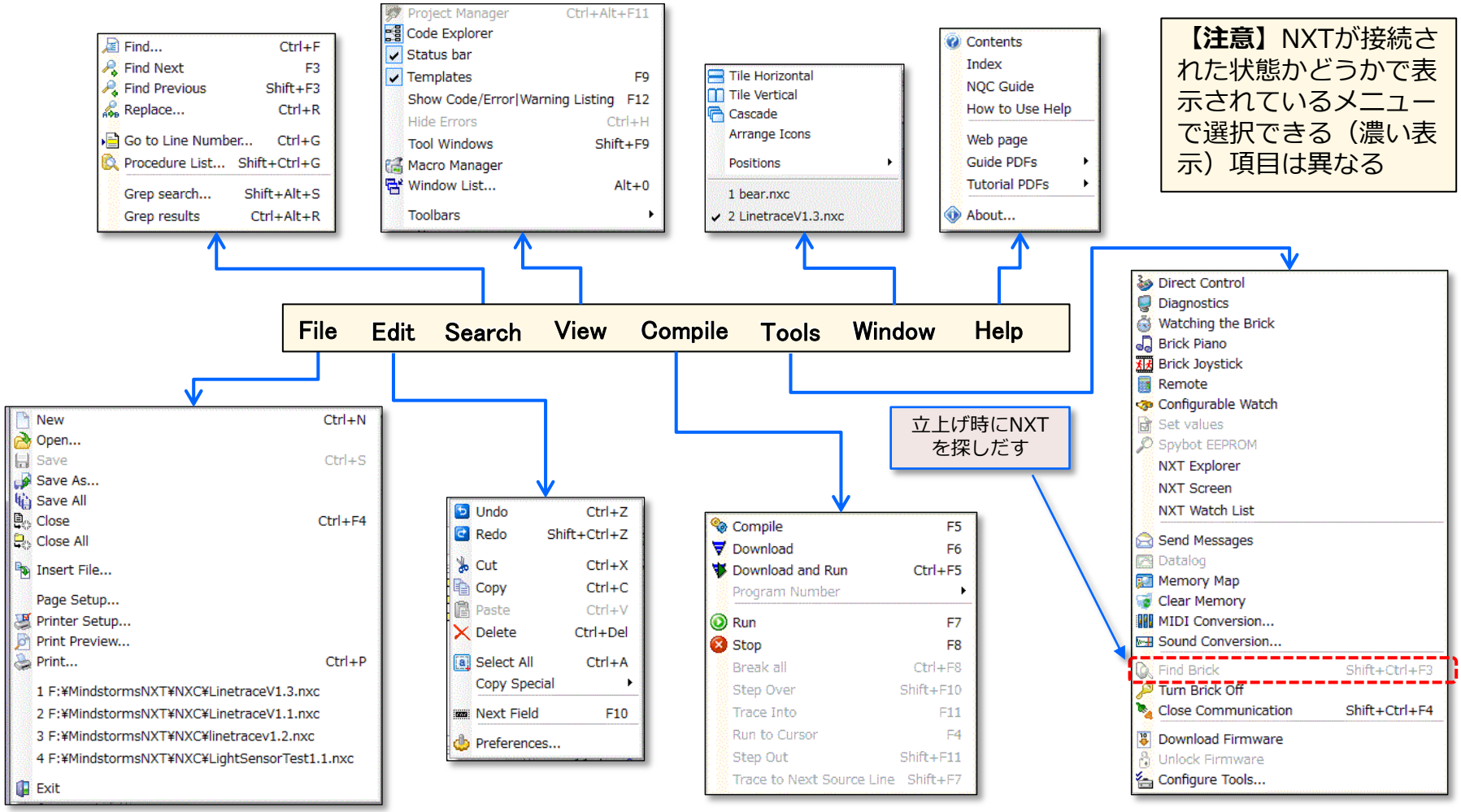
接続中の相手を表示

編集状態を表示

マクロ記録状態やエラー表示

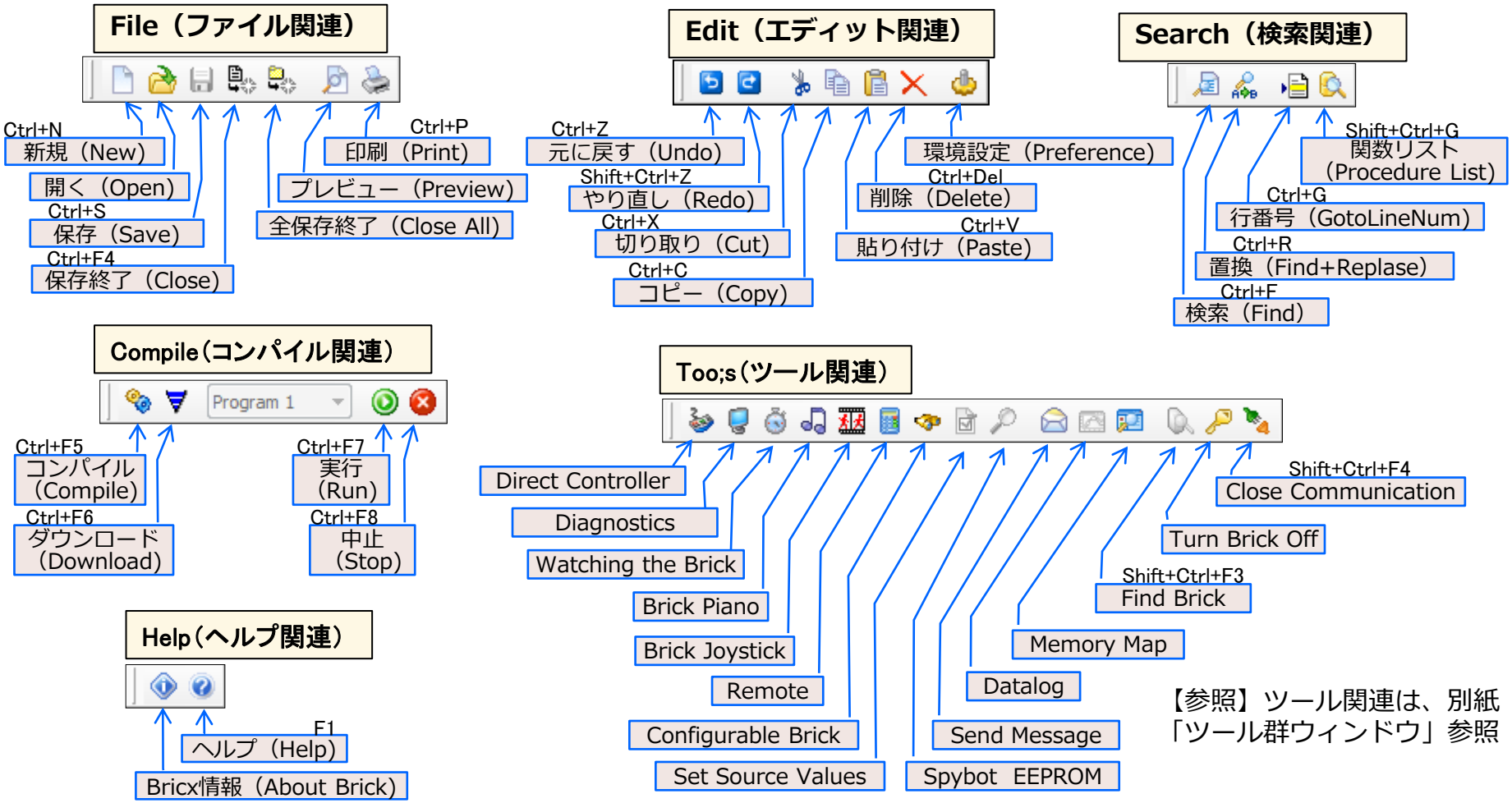
メニューバー

BricxCCのメニューバー（プルダウンメニュー）のメニュー内容は、以下のようになっている。



ツールバー（アイコンメニュー）

ツールバー（Toolbars）は、ツールバー「View」⇒「Toolbars」選択によって、アイコンメニューの表示On/Offを設定する。

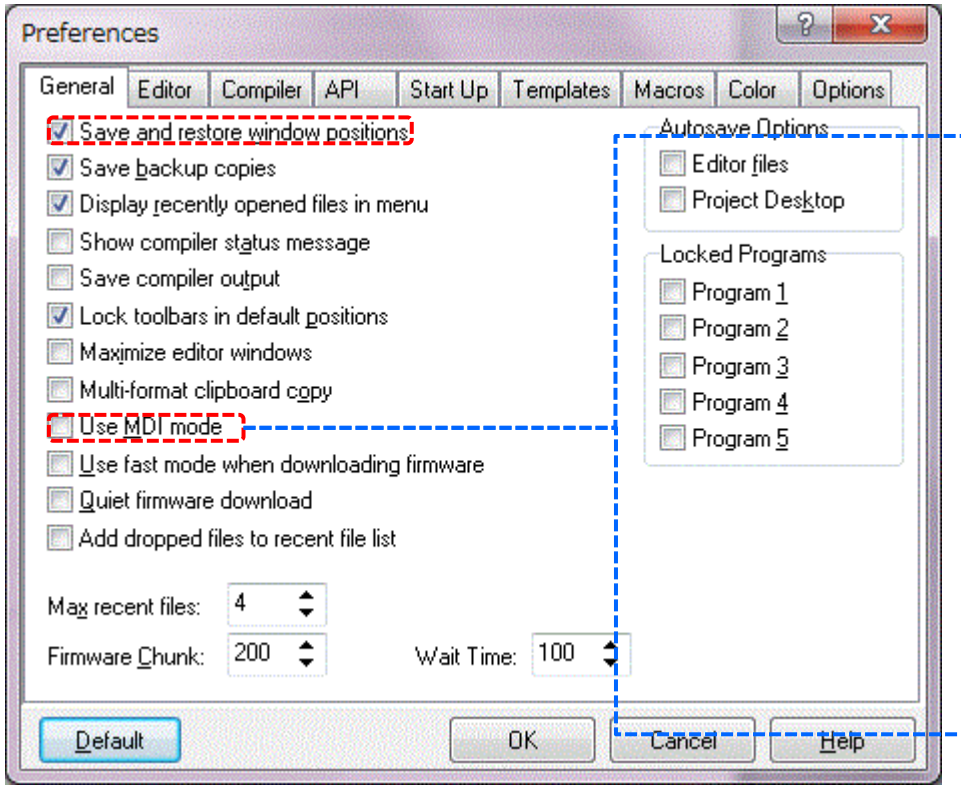


【参照】ツール関連は、別紙「ツール群ウィンドウ」参照

マルチ・エディタ画面の設定

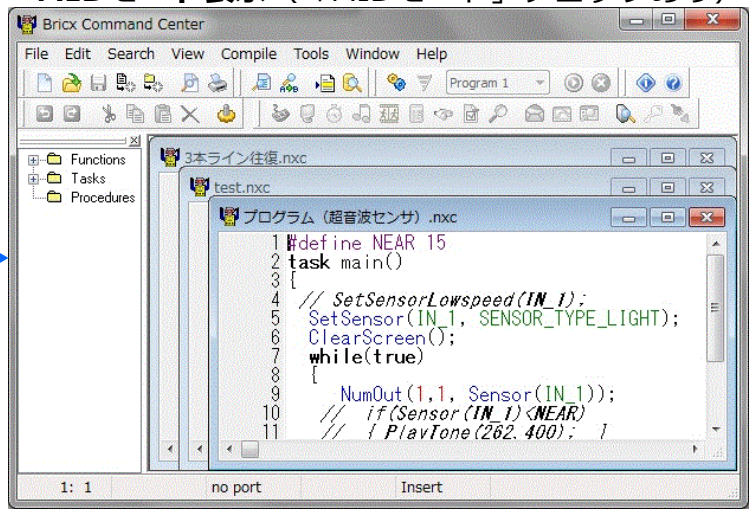
ツールバーの「Edit」⇒「Preferences」を選択して以下のGeneralタブを表示させる。

Preferences/General 画面

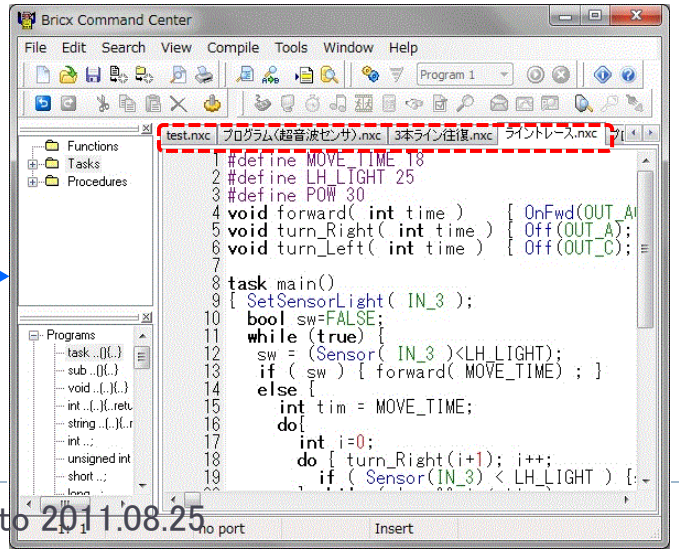


【注意】エディタ画面で複数編集する際の選択を行う。

MIDモード表示（「MIDモード」チェックあり）



タブ形式モード表示（「MIDモード」チェックなし）



立上げ省略設定

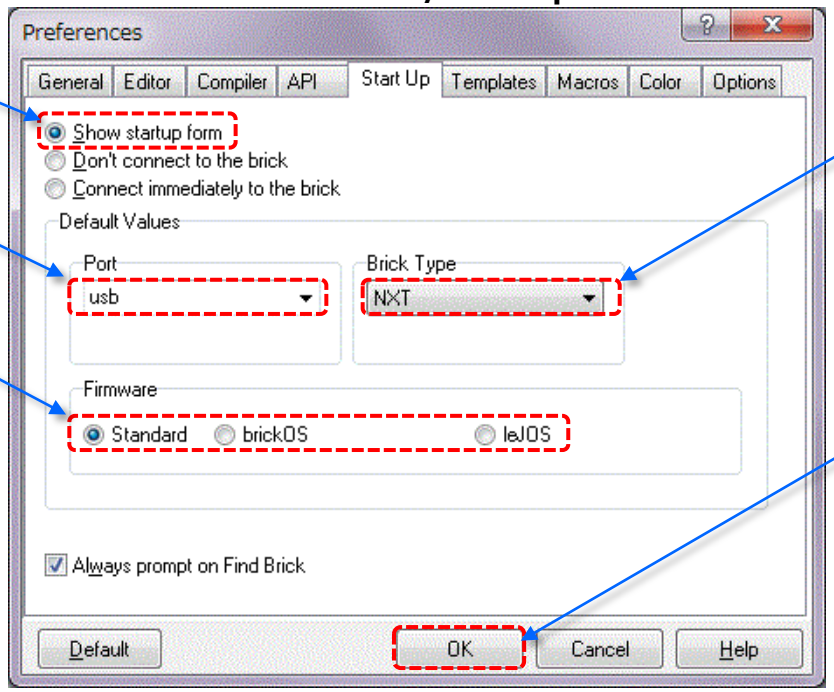
ツールバーの「Edit」⇒「Preferences」を選択して以下のStartUPタブを表示させる。

◆「Show ･･･」は、常に「Find Brick」画面を表示起動
◆「Don't ･･･」は、NXTと接続せず起動
◆「Connect ･･･」は、「Find Brick」画面を表示せず起動

「usb」または「Bluetooth」を選択

「Standard」を選択

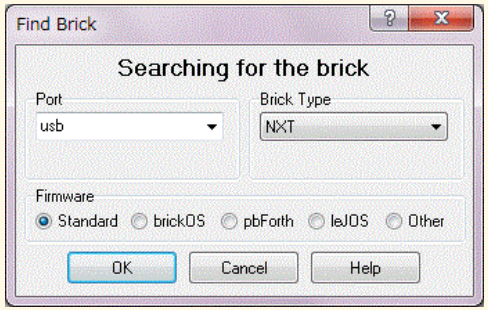
Preferences/StartUp 画面



「NXT」を選択

「OK」で終了

Find Brick画面



立上げ時のNXT接続画面

この設定をしておくことで、最初の立上げ画面「Find Brick」での省略選択が、ここでの設定値が表示される。何度も設定し直すよりも、ここで一度設定しておくことで立上げが早くなる。

コンパイル初期設定

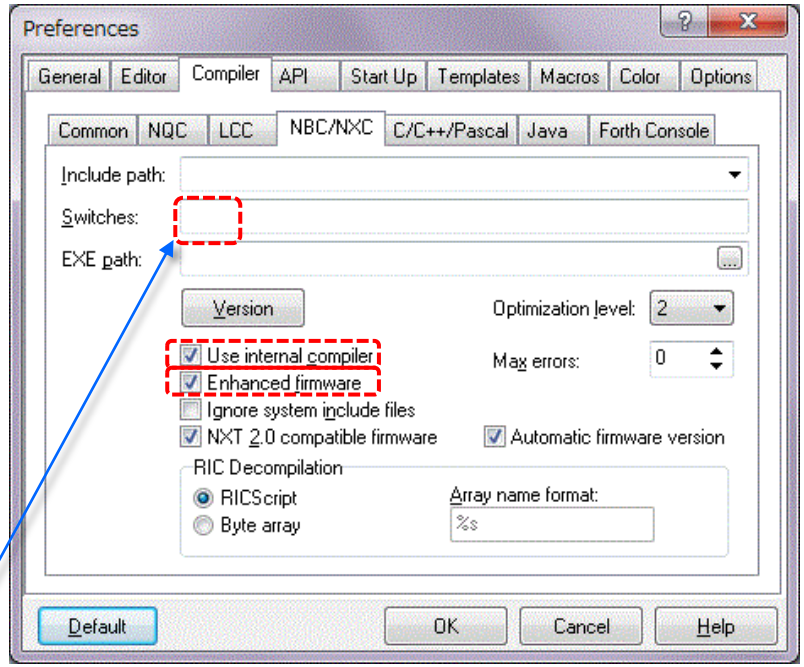
ツールバーの「Edit」⇒「Preferences」を選択して以下のCompiler/Commonタブを表示させる。

Preferences/Compiler/Common 画面

「－q」を設定することで、NXTにダウンロードしたときのビープ音を消去することができる。

ツールバーの「Edit」⇒「Preferences」⇒「Compiler」を選択して以下のNBC/NXCタブを表示させる。

Preferences/Compiler/NBC/NXC 画面

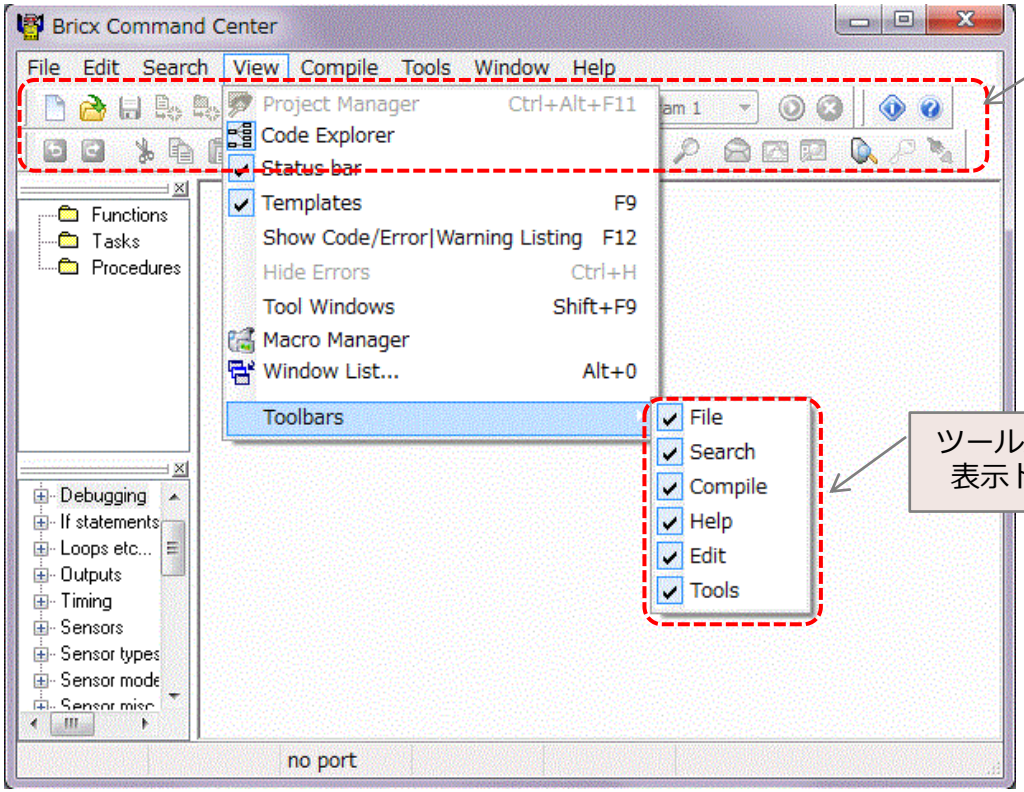


※「Use internal compiler」の設定によって、NXTへのダウンロード・スピードが驚くほど速くなる。
※「Enhanced firmware」の設定で、拡張ファームウェアを利用することを設定する。

ツールバー関連の表示オプション

BricxCC画面上のツールバーの表示オプションの切り替えを行う。

BricxCC画面上でのView/Toolbars設定メニュー



ツールバー

※ツールバーの中の機能を探すより、よく使う機能はツールバーにアイコンを表示させておくほうが効率アップできる。

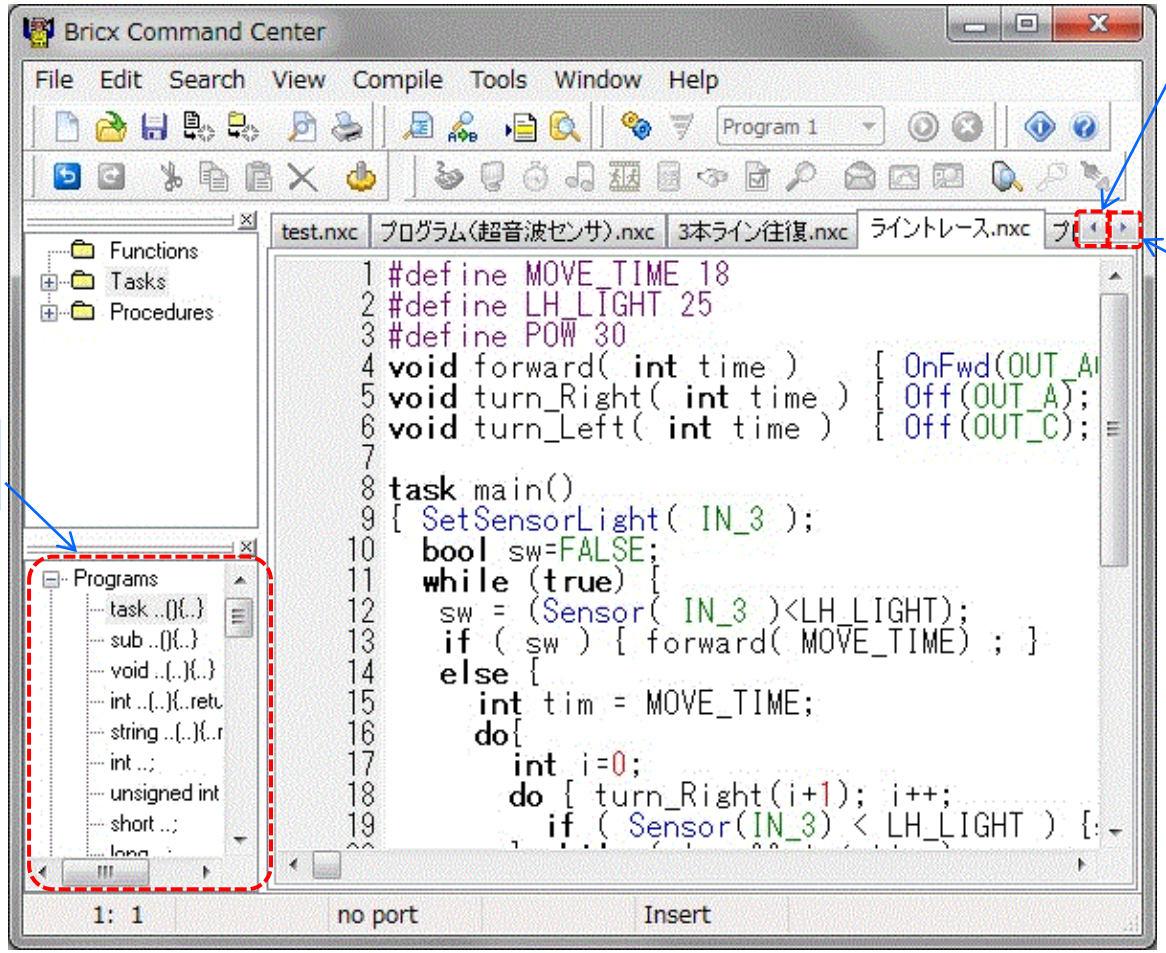
ツールバーの表示トグル

BricxCC画面のプルダインメニュー「View」⇒「Toolbars」を選択

プログラム・エディタ

プログラム・エディタ上の簡単操作を覚えることで、効率を高めることができる。

BricxCC画面上でのプログラム・エディタ関連情報



前ファイルへ
(Ctrl+Shift+Tab)

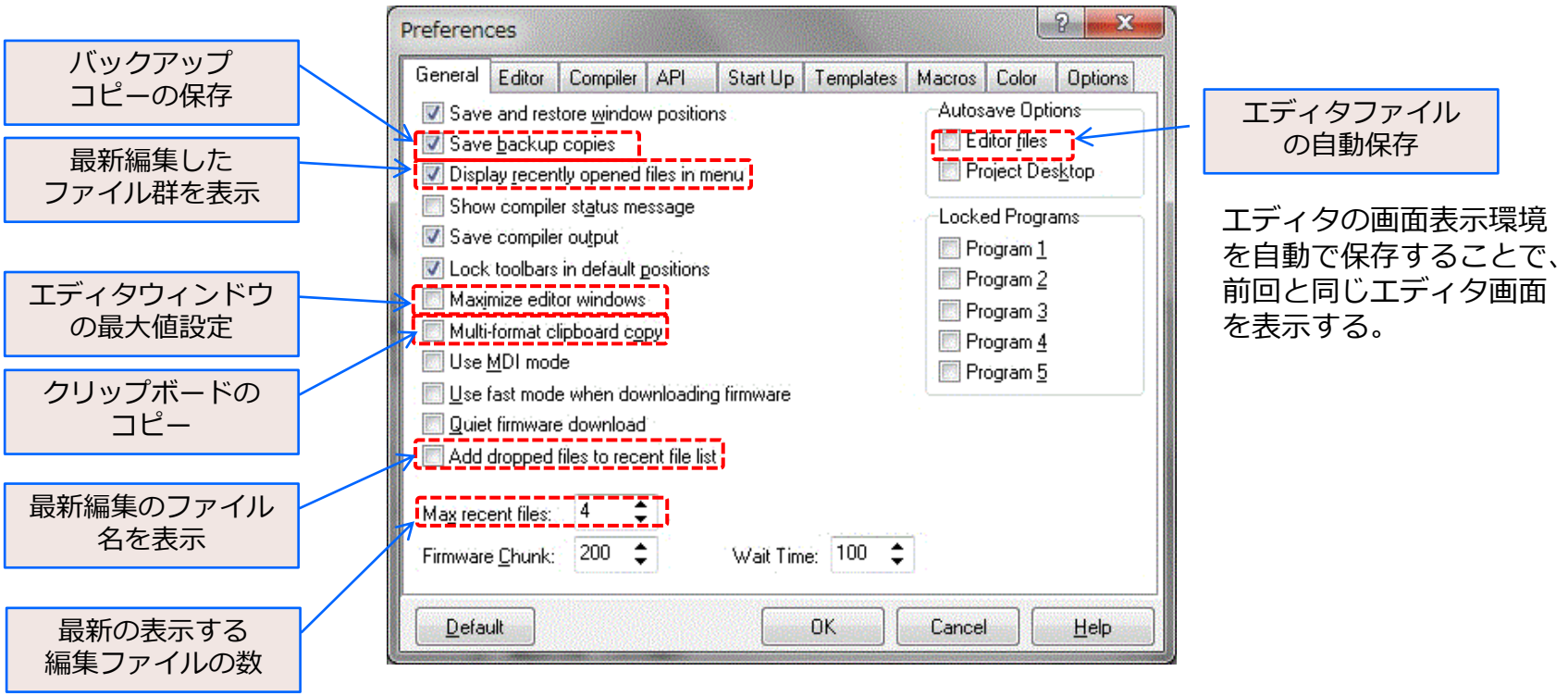
次ファイルへ
(Ctrl+Tab)

F9キーで
表示On/Off切替

プログラム・エディタの画面オプション

プログラム・エディタ上の簡単操作を覚えることで、効率を高めることができる。

Preferences/General 画面

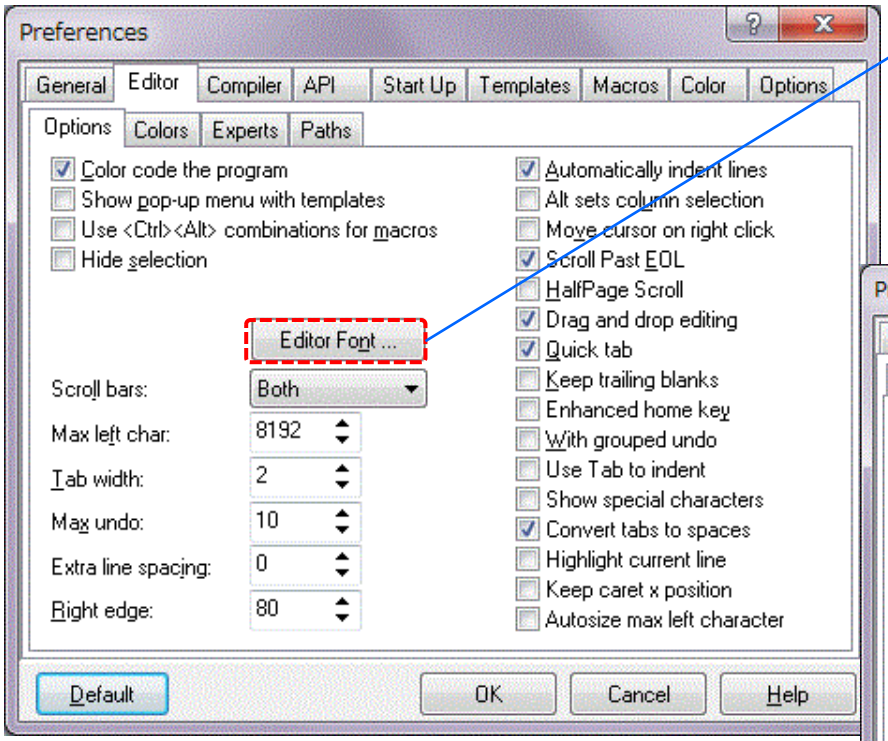


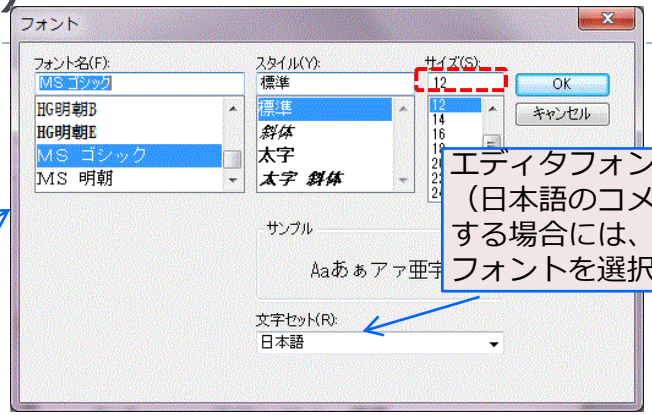
エディタの画面表示環境を自動で保存することで、前回と同じエディタ画面を表示する。

エディタ・ページの設定（１）

エディタ・ページで設定すべきオプション
ツールバーの「Preferences」⇒「Editor」を選択

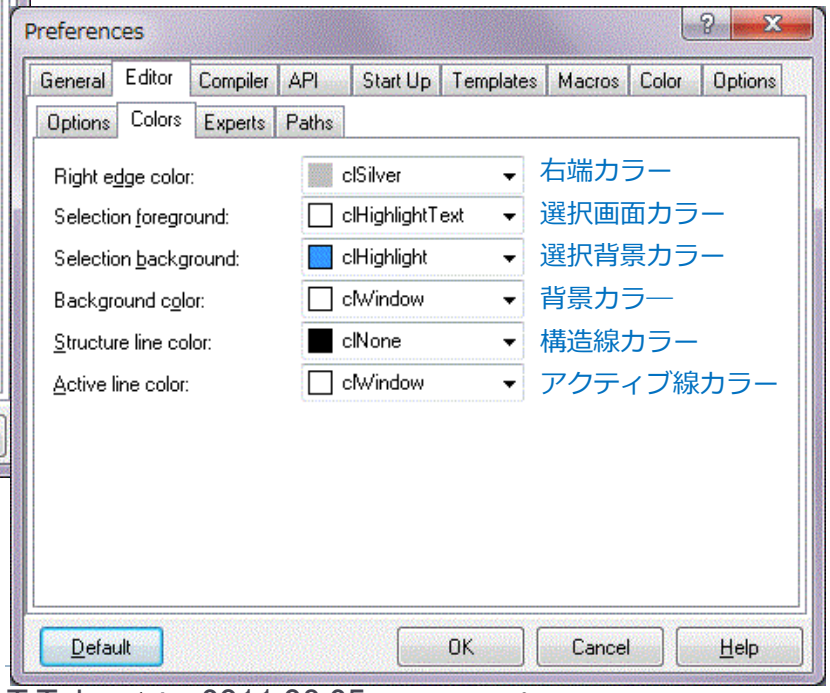
Preferences/Editor/Options画面





エディタフォントを設定
（日本語のコメントを記載する場合には、必ず日本語フォントを選択のこと）

Preferences/Editor/Colors画面

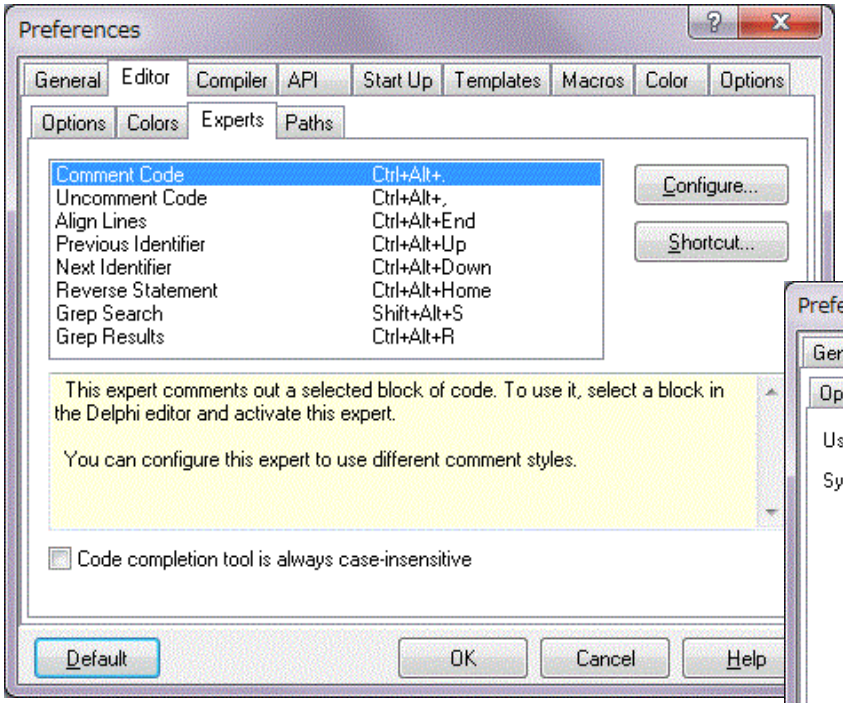


※ 日本語のコメントを入れる場合には設定必須
文字フォント・スタイル・サイズを設定する。

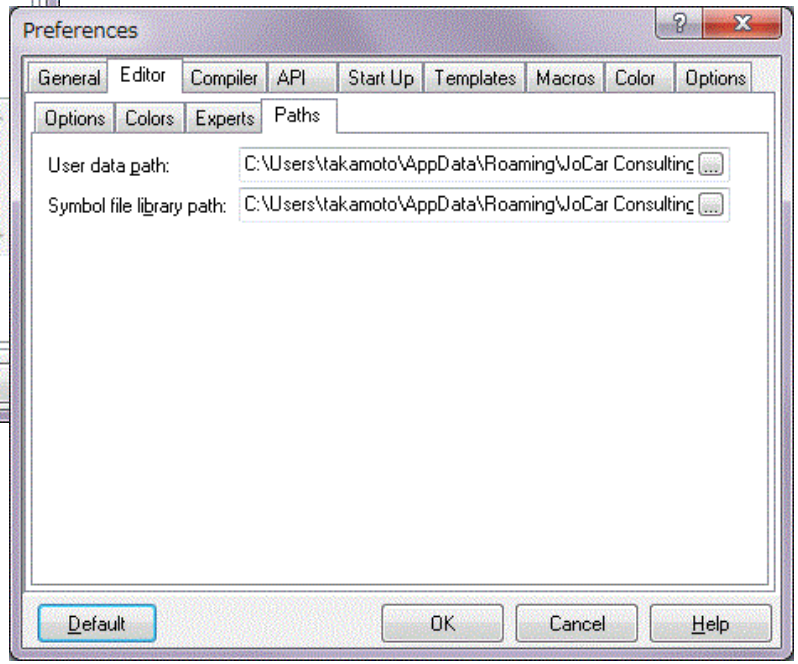
エディタ・ページの設定（２）

エディタ・ページで設定すべきオプション関連となる。

Preferences/Editor/Experts画面



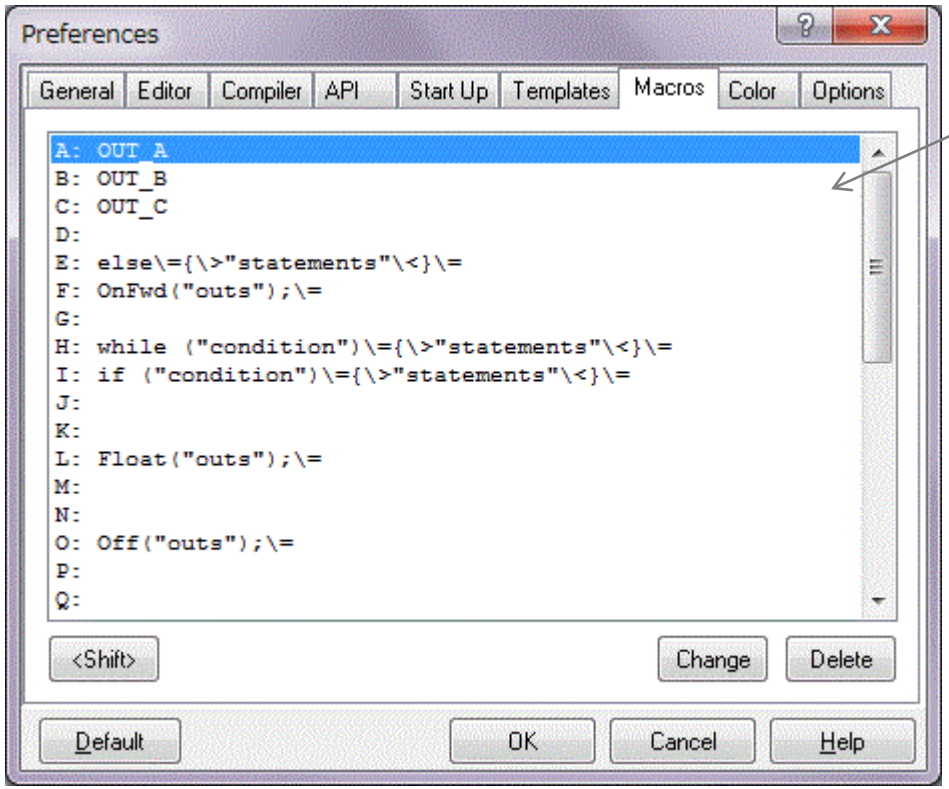
Preferences/Editor/Paths画面



マクロ機能

マクロ機能は、「Preferences」→「Macros」から選択する。
マクロは、エディタ上で、「Ctrl」＋「Alt」＋＜マクロキー＞、もしくは、「Shift]＋「Ctrl」＋「Alt」＋＜マクロキー＞を入力して利用する。
＜マクロキー＞は、A～Zと０～９までとなる。

Preferences/Macros 画面



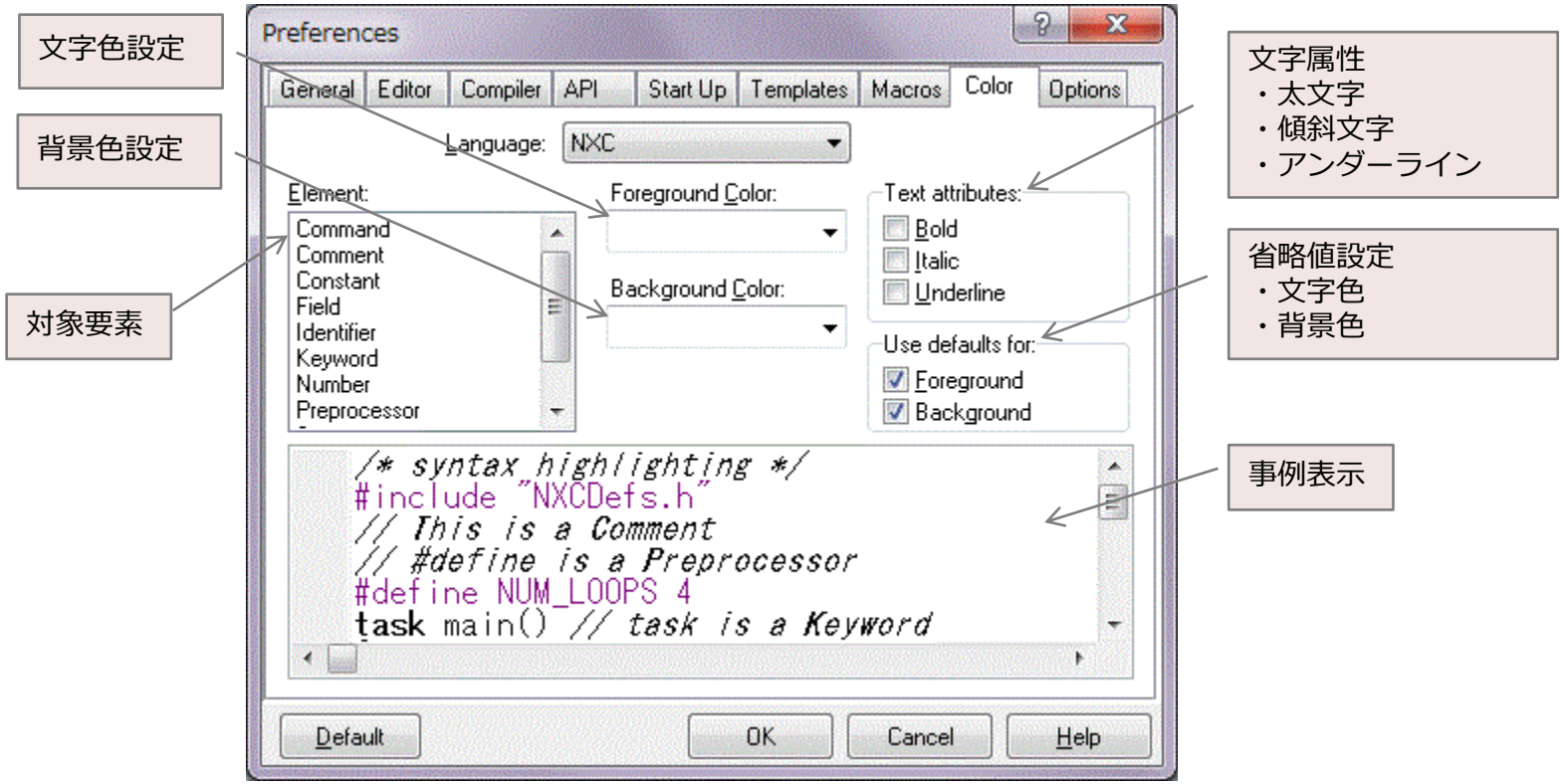
＜Ctrl＞＋＜Alt＞で起動するマクロ設定（既存設定）

（注意）あらかじめ環境設定「General」の左側３行目の「Use <Ctrl><Alt>combinations for macros」にチェックを入れておく必要がある。チェックが無いと、マクロは起動しない。

表示カラー設定

表示カラー設定機能は、「Preferences」→「Color」から選択する。
この設定で、エディタ上に表示されるソースコードを色分けで見やすくすることができる。各対象となる**要素**ごとに設定する。

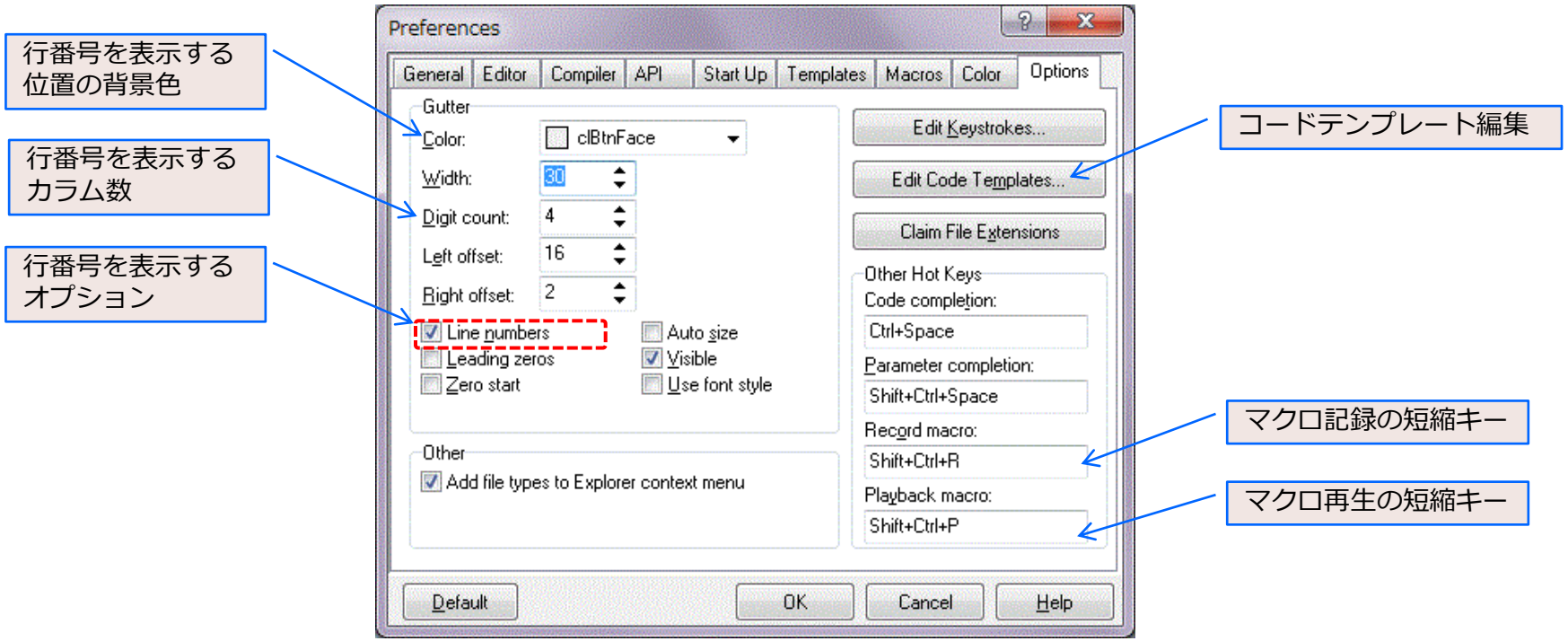
Preferences/Color 画面 （NXC関連設定）



エディタ・オプション設定

環境設定のその他として、「Options」タブから、エディタ関連のオプション機能の編集を行う。

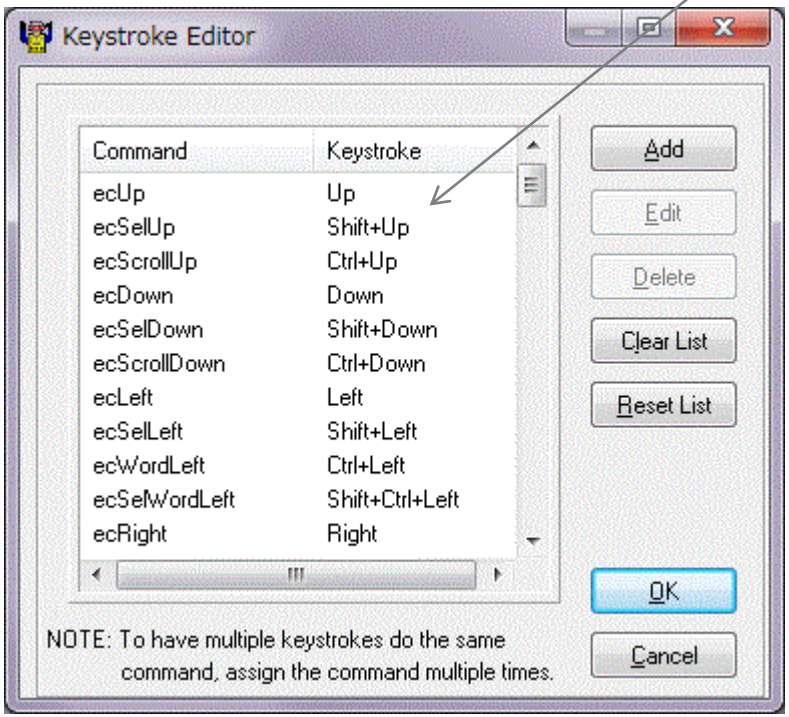
Preferences/Options 画面



キー割り当て設定

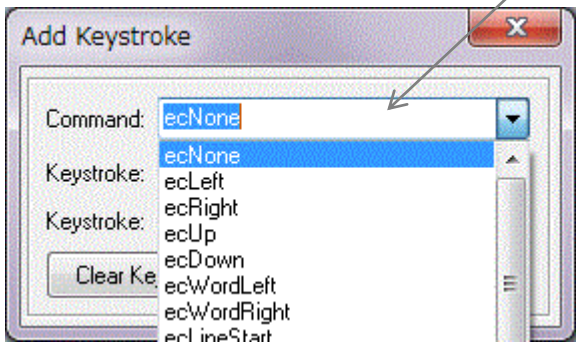
キーボード入力によるコマンドの割り付けを行う

Keystroke Editor画面



キーボードの追加画面

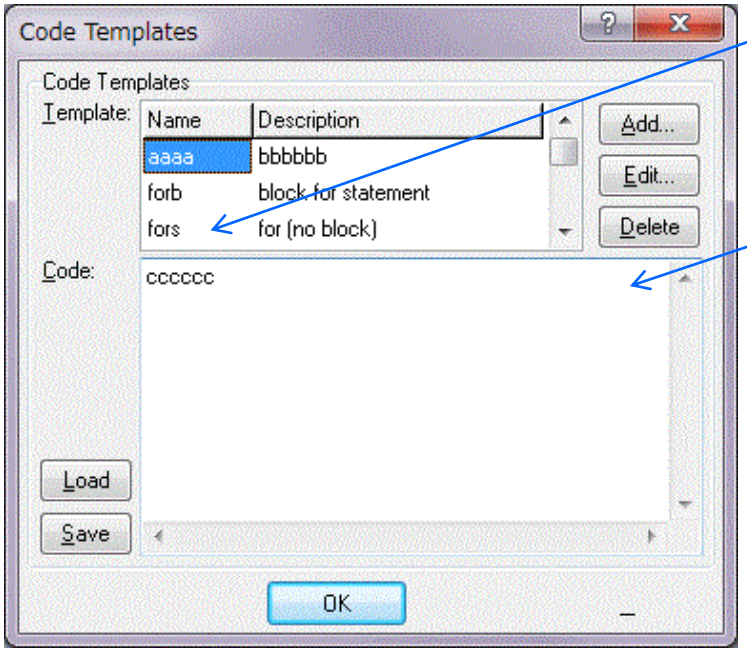
Add Keystroke 画面



コードテンプレート設定

コードテンプレートは、ユーザ設定によるテンプレート展開を、エディタ編集画面上のカーソル位置で <Ctrl> + <J> で行うことができる。

Code Templates 画面



エディタ編集中であらかじめ記入しておく名前（Name)となる。

ここで編集を行う

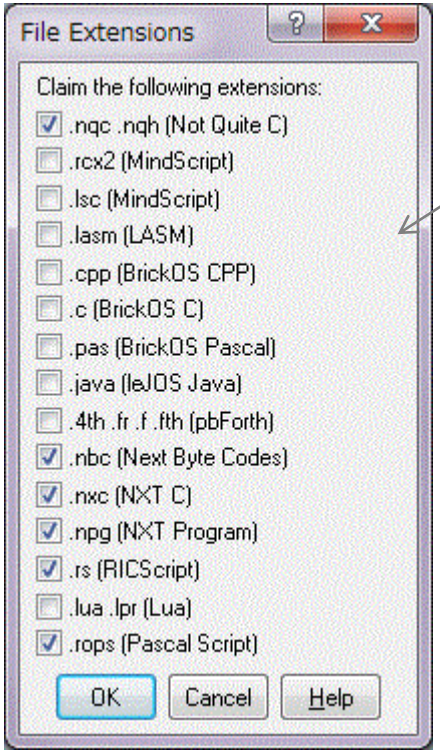
（注意）Codeで編集する際、「｜」を入れることで、テンプレート展開時に、カーソル位置を設定できる。

事例として、Name を「sets」として、
Codeを「 SetSensor(　｜,);」と作成するし、
エディタ画面で、テンプレートを展開するカーソル位置で、 <Ctrl> + <J> を入力しることで、テンプレート展開する。

ファイル拡張子認識

BricxCCで識別対象となるファイル拡張子の設定となる。
ツールバーの「Tools」の「NXT Explorer」で、認識対象となるファイルとなる。

File Extensions 画面

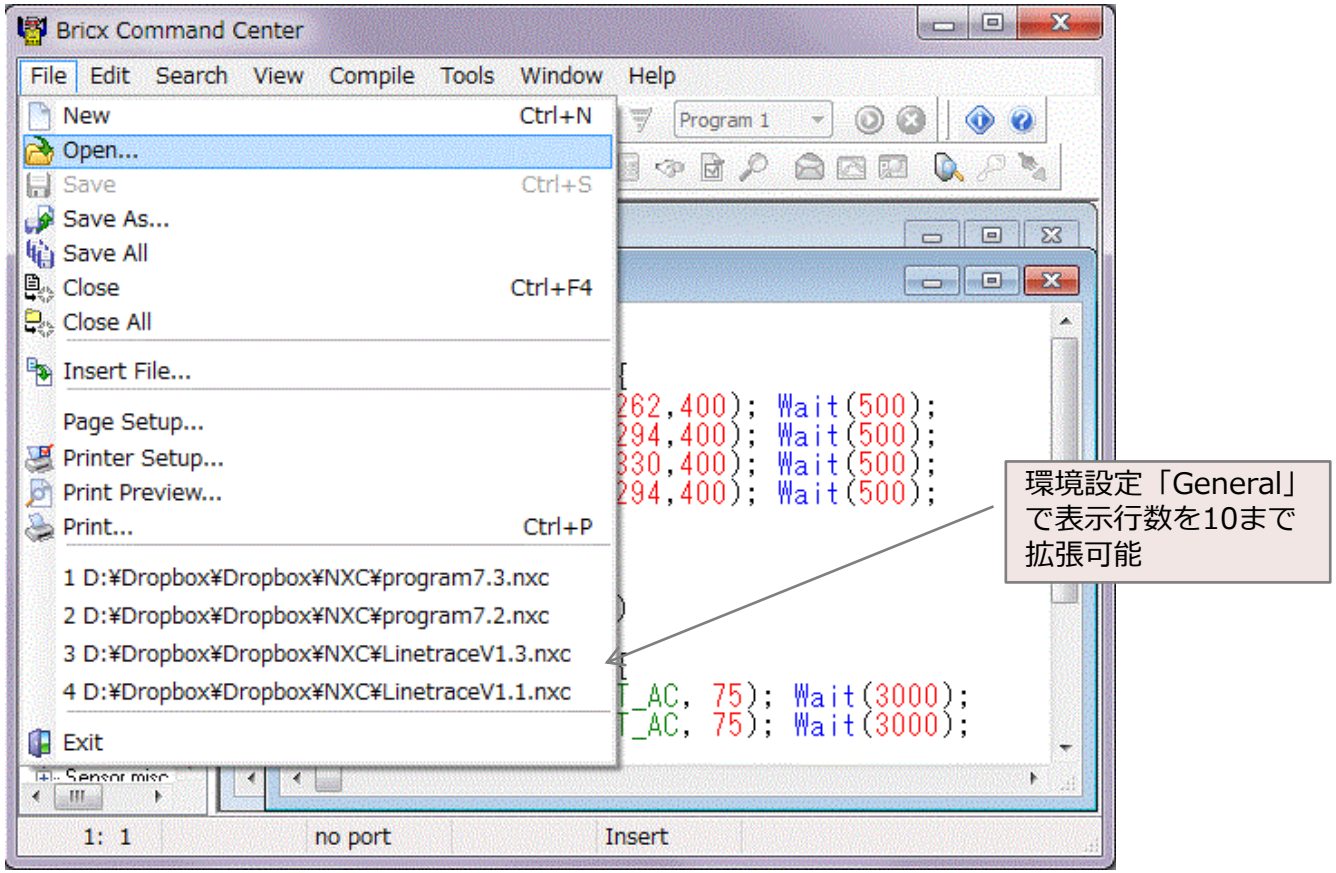


ファイル拡張子の識別を行うチェック欄

ファイル・メニューオプション

メニューバー「File]の一覧表となる。主にファイルの一般的な新規作成、オープン、クローズ、それにプリント出力関係などが用意されている。

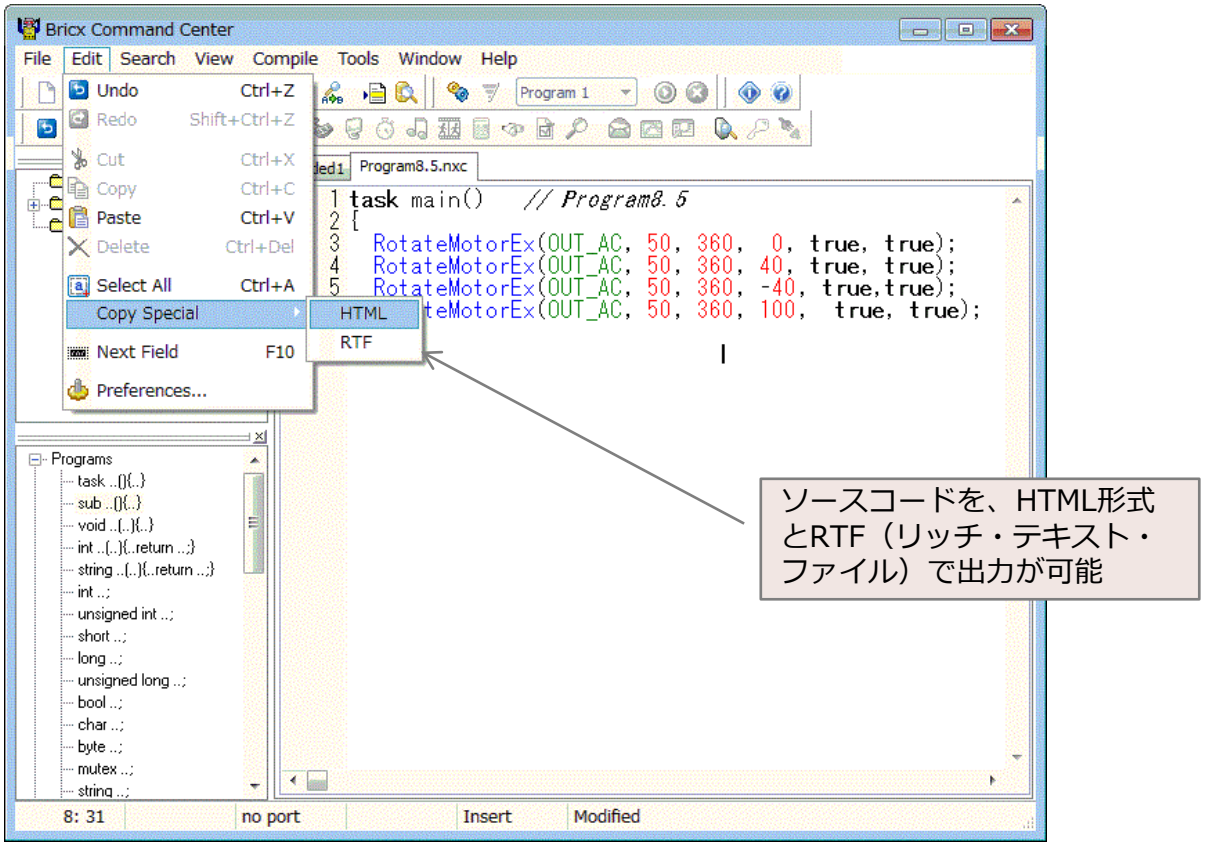
ファイル メニュー



エディタ・メニューオプション

メニューバー「Edit]の一覧表となる。この中にある「Copy Special」には、
２つのファイル・フォーマット（HTMLかRTF）で出力することができる。

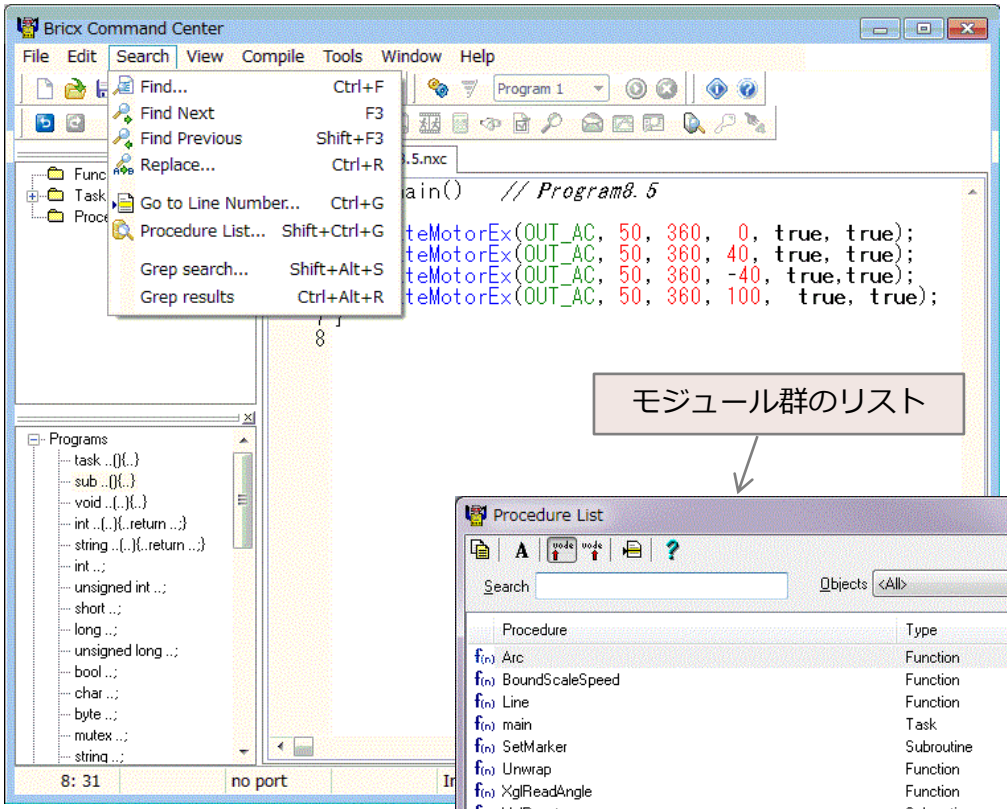
エディト/コピー特殊メニュー



検索メニューとツールバー

メニューバー「Search」の内容表示。

検索メニュー

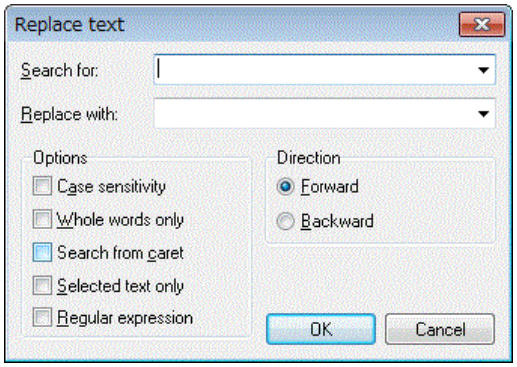


モジュール群のリスト

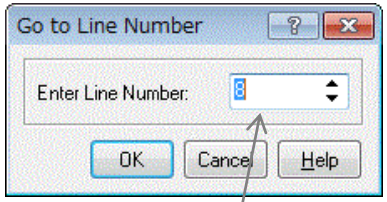
Procedure List 画面

文字列の置換を行う画面

Replace text 画面



Go to Line Number 画面



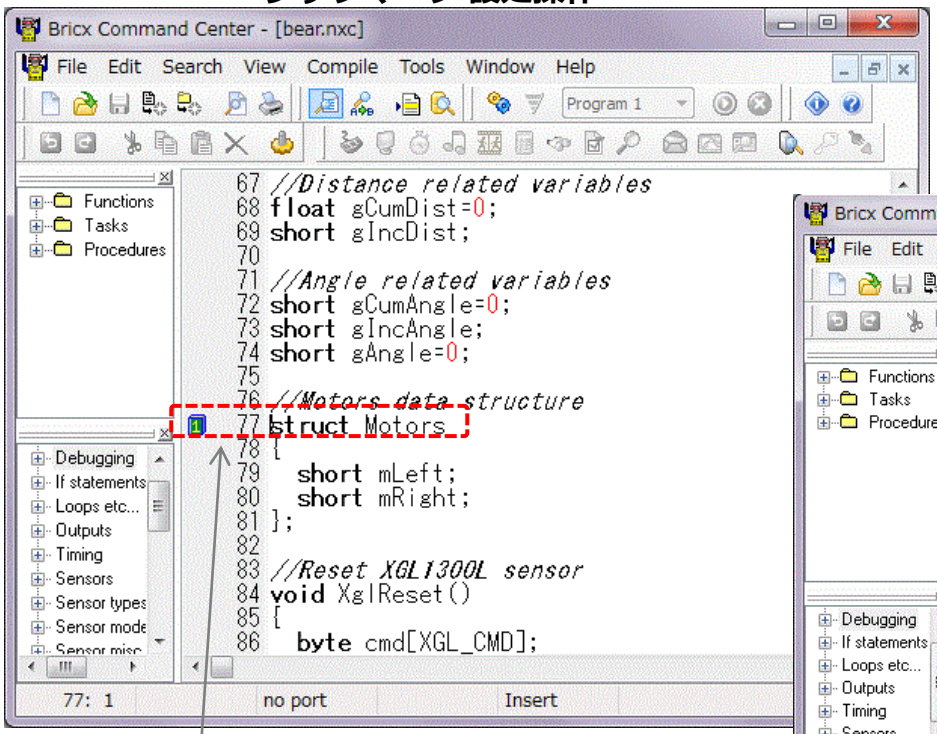
行番号へのジャンプ

ブックマーク機能

エディタのライン上で「**Shift＋Ctrl＋<数字>**」で、ブックマークをライン上に表示させることができる。ここで、数字は、0～9まで。ブックマークへの即時の切り替えが可能となる。

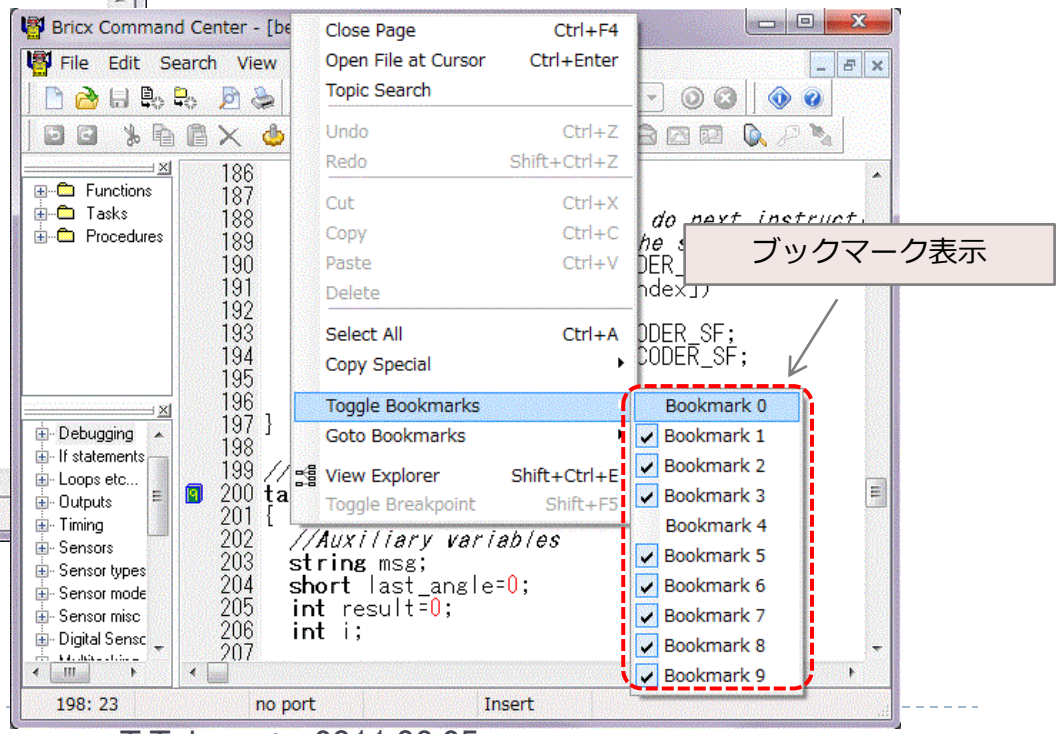
記憶させたブックマークを呼び出し、ジャンプすることができる。
この場合、エディタ上でマウス右ボタンを押して、以下のメニューを表示させて、ブックマークを選択して即ジャンプする。

ブックマーク 設定操作



ブックマーク設定

ブックマーク・呼び出し画面

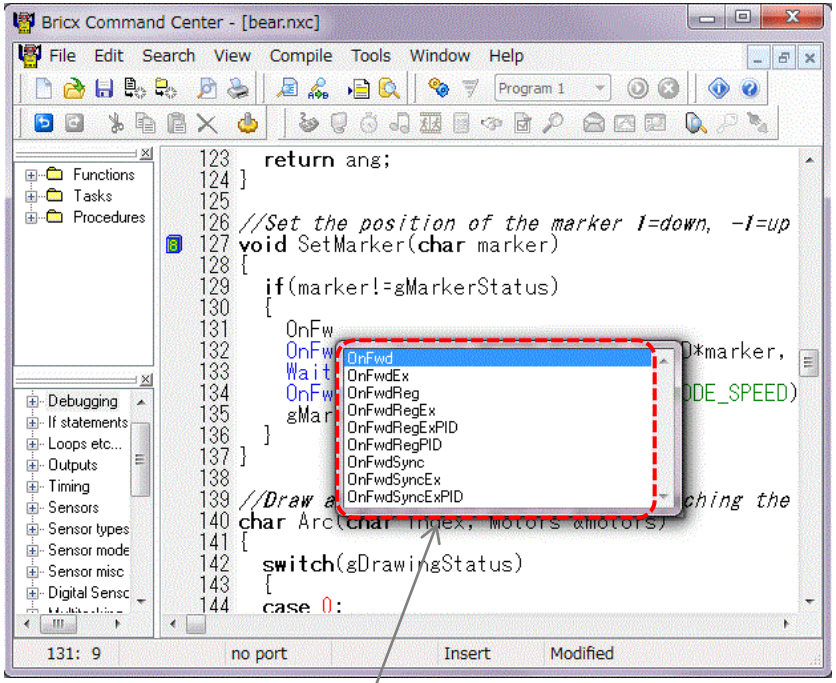


ブックマーク表示

エディタ編集操作補助機能

エディタ中で、コードを途中まで入力した段階で、
「Ctrl＋Space」キーを押すと、候補対象のコードが
表示される。

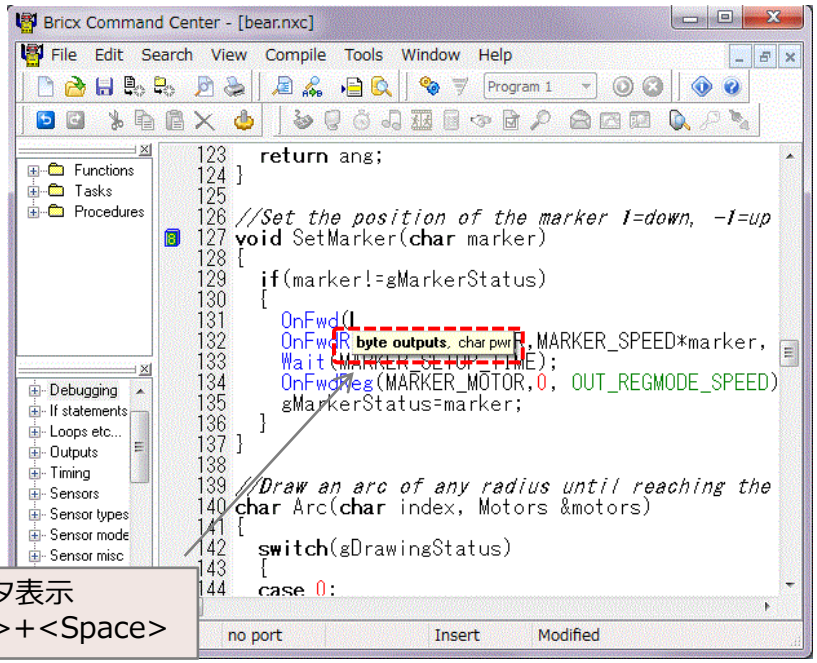
コード候補補助機能



コード候補
<Ctrl>+<Space>

エディタ中で、関数コードを入力した段階で、パラメータ（引数）が分からない場合に、**「Shift+Ctrl＋Space」キー**を押すと、パラメータのヒントが表示される。

パラメータ表示補助機能

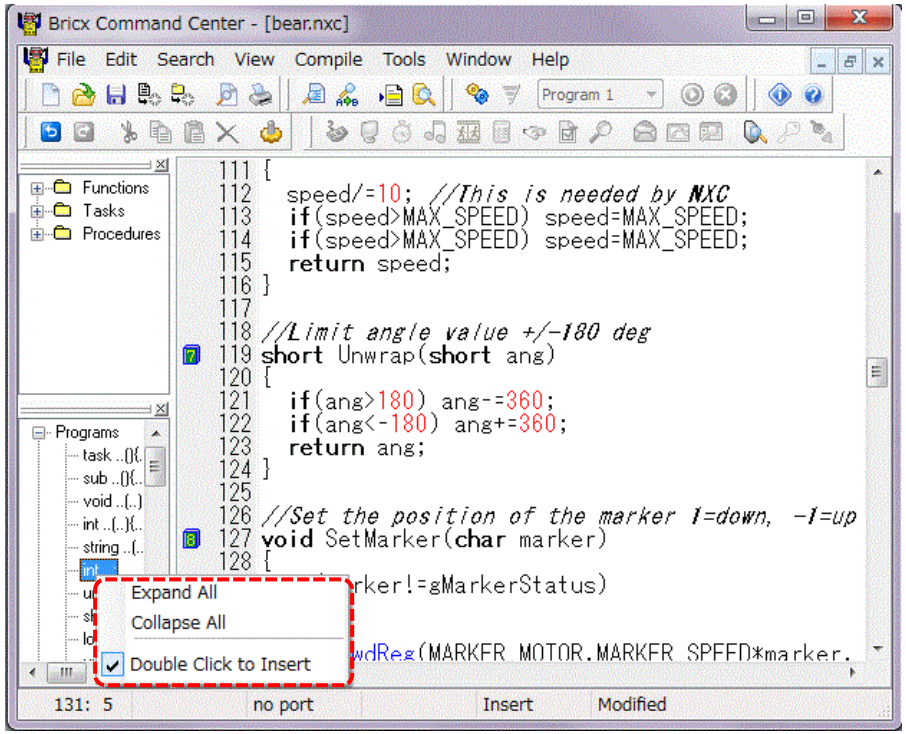


パラメータ表示
<Shift>+<Ctrl>+<Space>

ドラッグ・ドロップ・プログラミング

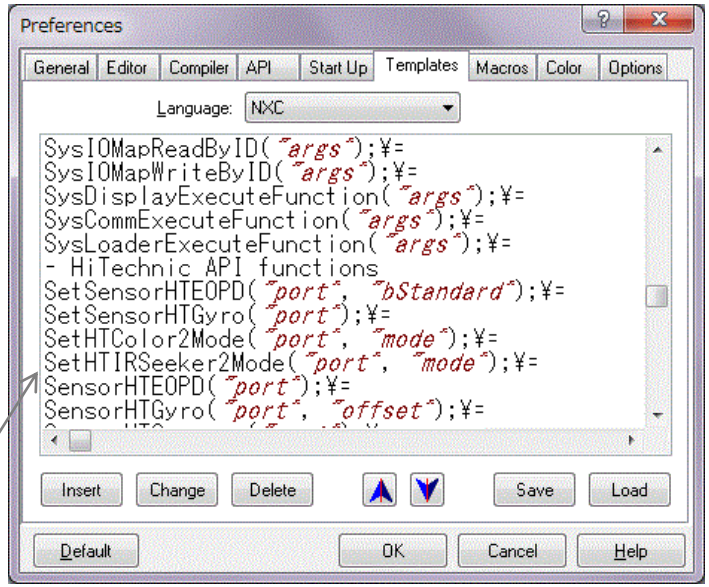
テンプレートウィンドウから選択したコードを、ダブルクリックすることで、エディタ上にプログラムコードを展開することができる。

テンプレート展開機能



ツールバーの「**Edit**」→「**Preferences**」→「**Templates**」の画面操作によって、マクロのテンプレートを編集することができる。

テンプレート編集画面



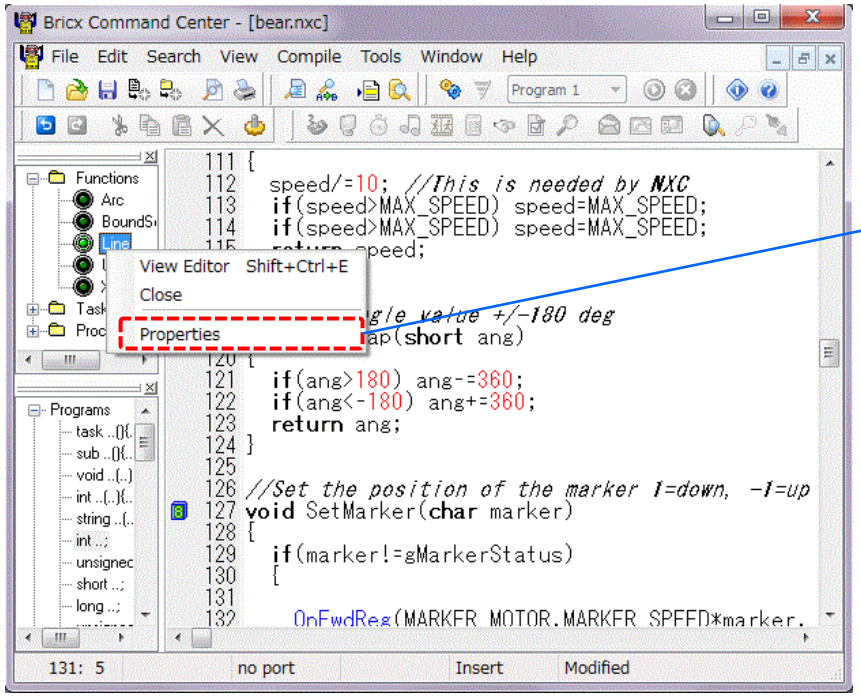
テンプレート内容表示

コード・エクスプローラ画面

コード・エクスプローラ表示、およびそのオプション表示
モジュール一覧を見ることができ、表示ジャンプできる。

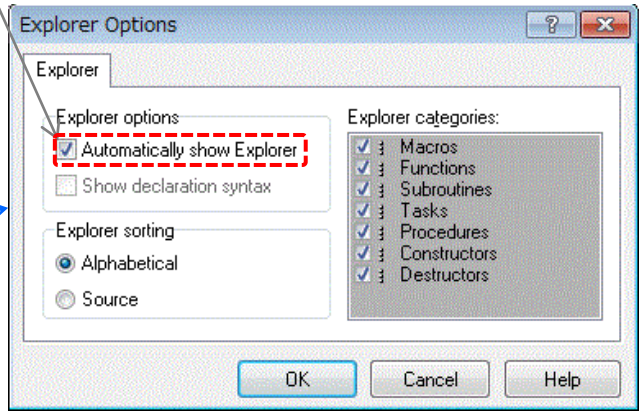
エクスプローラ表示
チェックを入れると
常に表示Onにできる

コード・エクスプローラ画面



このコード・エクスプローラ画面には、**関数**（Function）、**タスク**（Task）、および**プロシジャ**<サブルーチン>（Procedure）のモジュール一覧表を示し、ダブルクリックで、画面ジャンプする。

エクスプローラ オプション 画面



エクスプローラ表示カテゴリ
- ☐ マクロ（Macros）
- ☐ 関数（Functions）
- ☐ サブルーチン（Subroutines）
- ☐ タスク（Tasks）
- ☐ プロシジャ（Procedures）
- ☐ コンストラクタ（Constructors）
- ☐ デストラクタ（Destructors）

※NXCでは、コンストラクタとデストラクタは未使用

マクロ管理

マクロ管理画面は、ツールバーの「View」→「Macro Manager」で画面表示させることができる。

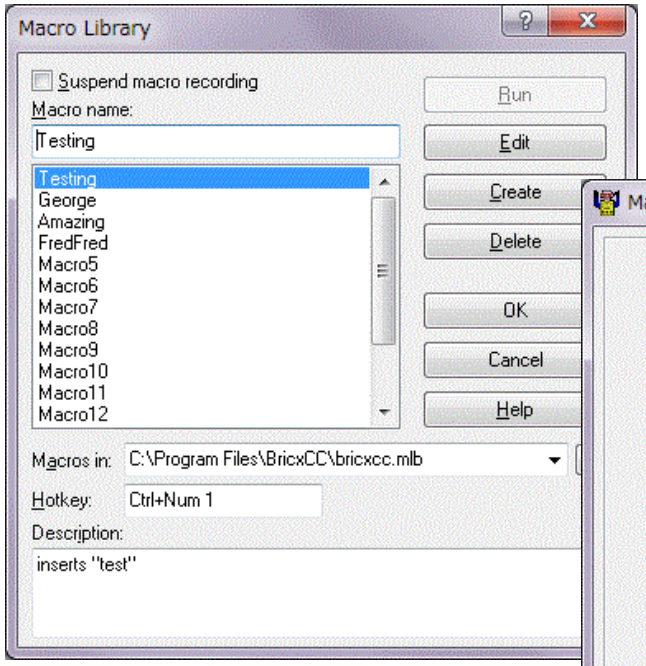
簡易マクロ定義

Shift + Ctrl + R
<マクロ操作>
Shift + Ctrl + R

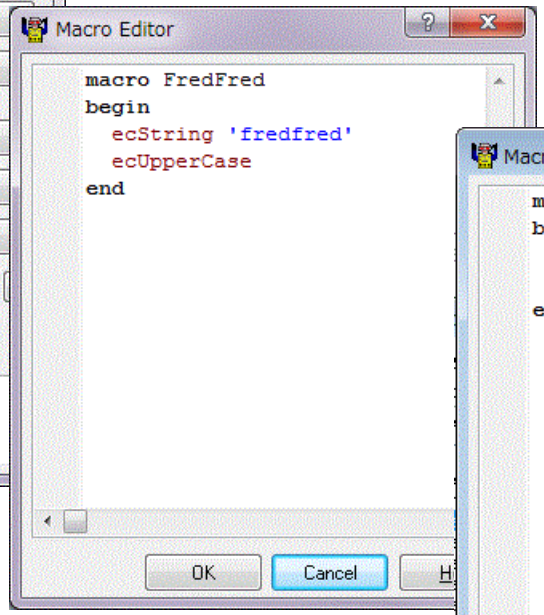
簡易マクロ実行

Shift + Ctrl + P

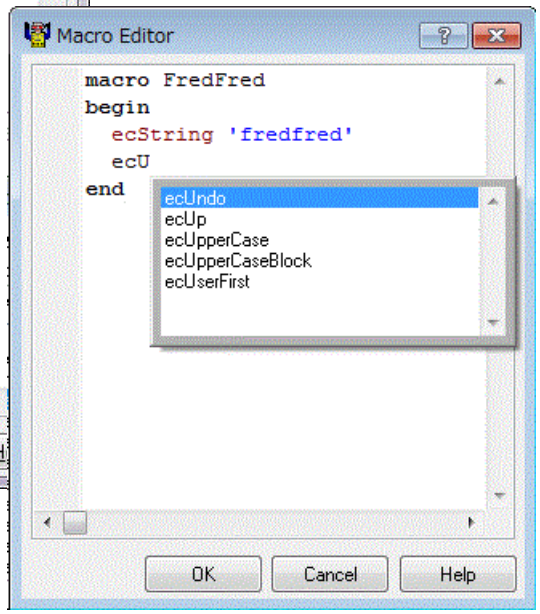
マクロ ライブラリ画面



マクロ エディタ画面（1）



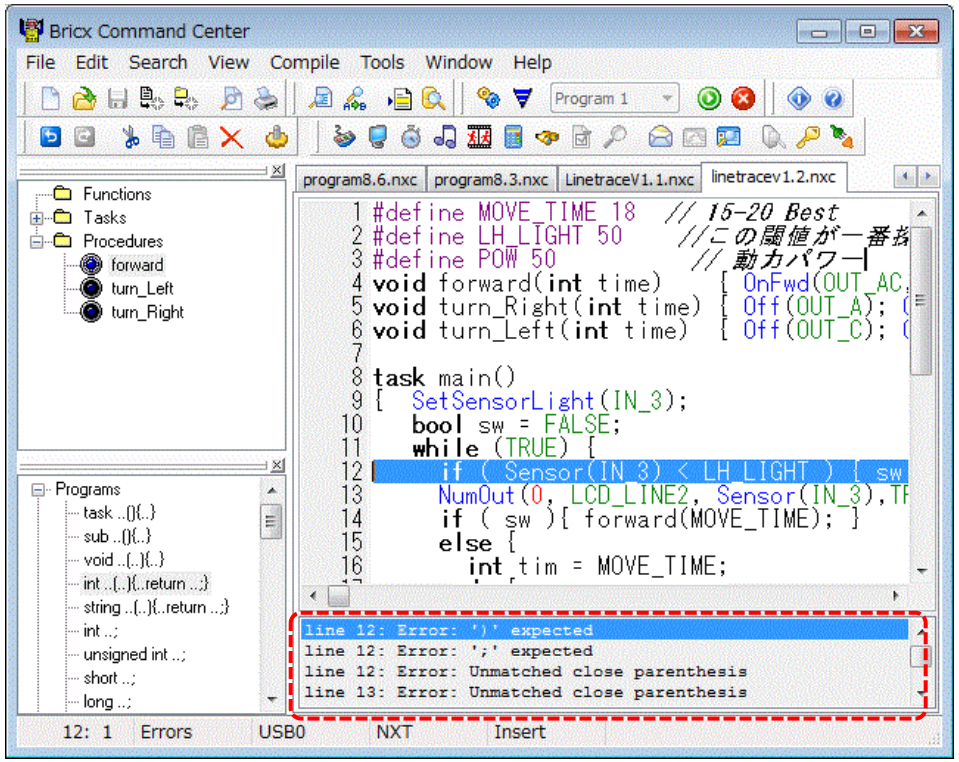
マクロ エディタ画面（2）



コンパイル・エラー表示機能

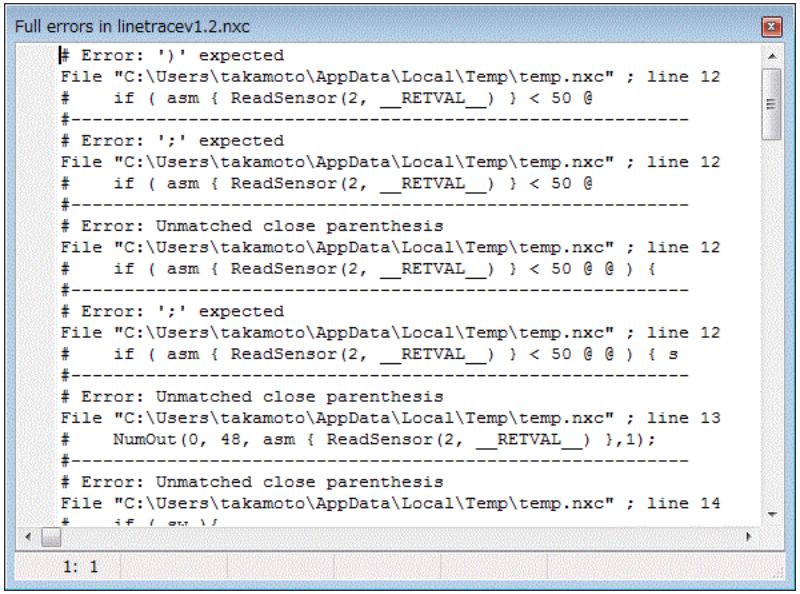
ツールバーの「Compiler」→「Compile」でエラーが出た場合には、以下のようにエディタ画面下に行番号と一緒にエラー内容が表示される。

コンパイル・エラー表示 画面



エラー状態で、**F12キー**を押すことで、詳細なエラー内容が別ウィンドウで表示される。

コンパイル・エラー詳細表示 画面



ツール群ウィンドウ（１）

ツールバーの「Tools」には、多くの便利な機能が揃っている。ただし、これらが使える状態になるには、NXTが接続されていることが前提となる。

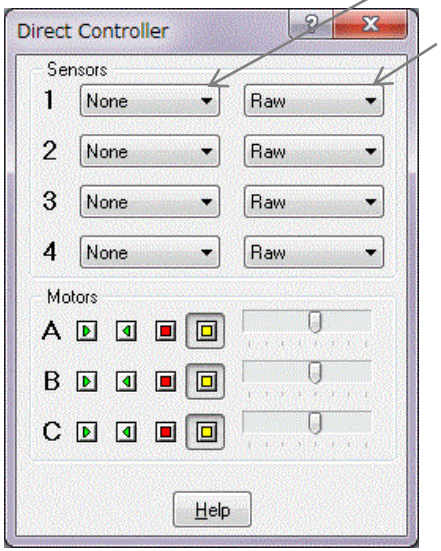
NXTが接続されていない場合には、ツールバーの「Find Brick」から、USBもしくはBloutoothされたNXTとを接続する必要がある。

ツール群 メニュー一覧

ツール群ウィンドウ（2）

ダイレクト・コントローラは、直接NXTのアクチュエータ（モータ）やセンサーを操作することができるウィンドウとなる。

Direct Controller



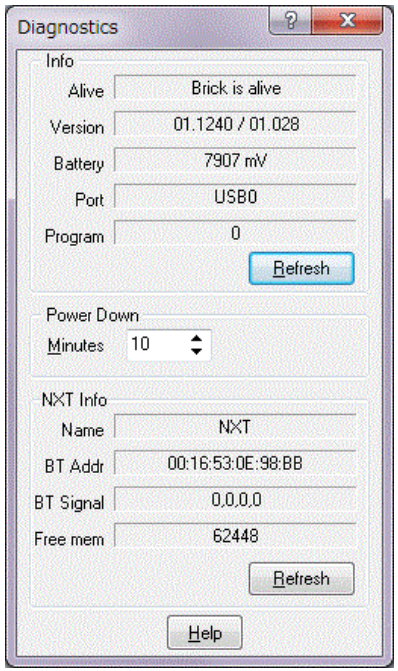
接続センサー種類を選択

読込データ種類を選択

【注意】センサー種類とデータ種類を設定しておくことで、次頁での「Watching the Brick」における「Data Analysis]のセンサーグラフが表示することが可能。

ダイアゴノスティックは、現状接続されているNXTの関連情報、例えばNXTの名前、ファームウェアのバージョン情報、バッテリー情報などを表示する。

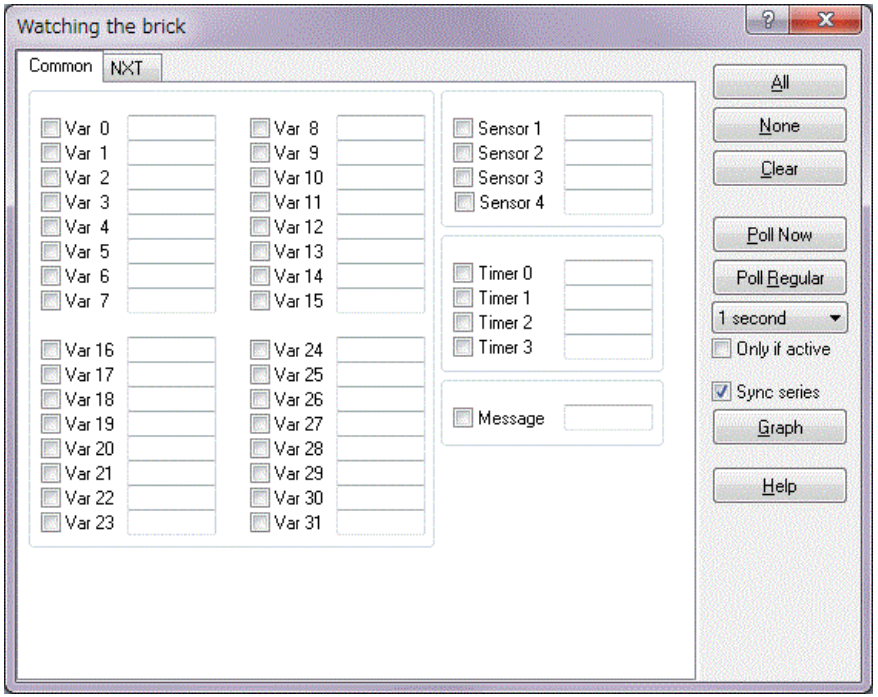
Diagnostics



ツール群ウィンドウ（３）

NXTのブロックツールで、各種センサーやその他値を時系列で値表示させたり、グラフ表示させたりすることができる。

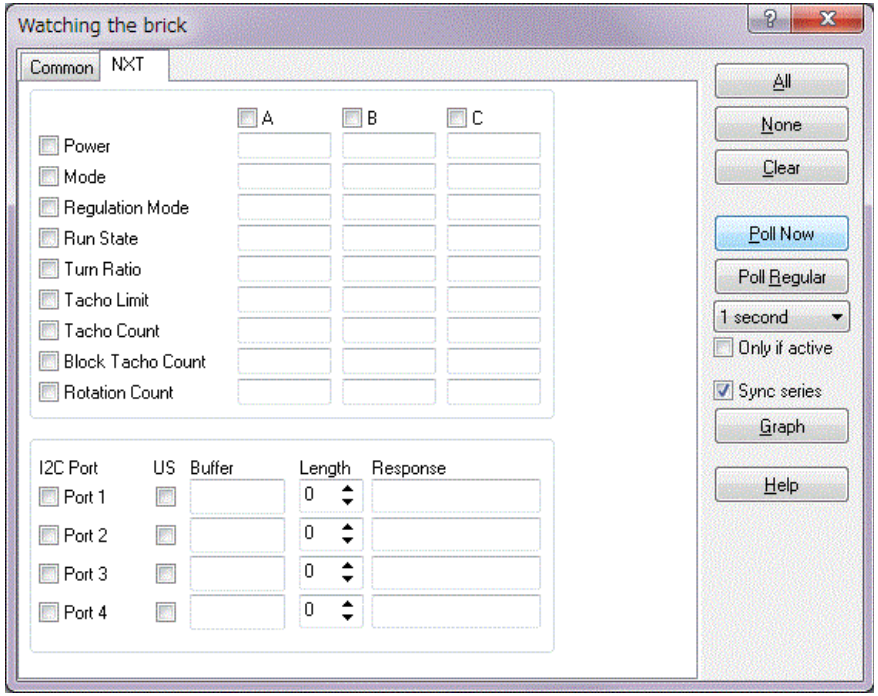
Watching the brick/Common 画面



ダイレクトコントローラのセンサー設定での読込を行う。センサー種類とデータ種類を設定した上で、センサー値を表示、さらにグラフ表示も可能。

NXTのブロックツールで、各種センサーやその他値を時系列で値表示させたり、グラフ表示させたりすることができる。

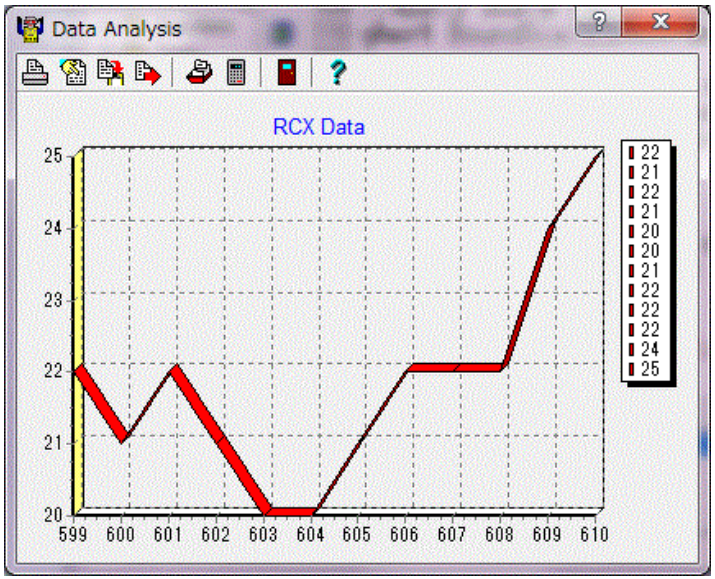
Watching the brick/ NXT 画面



ツール群ウィンドウ（４）

BricxCC上で、NXTのセンサー値などを動的に
グラフ表示させることができる。

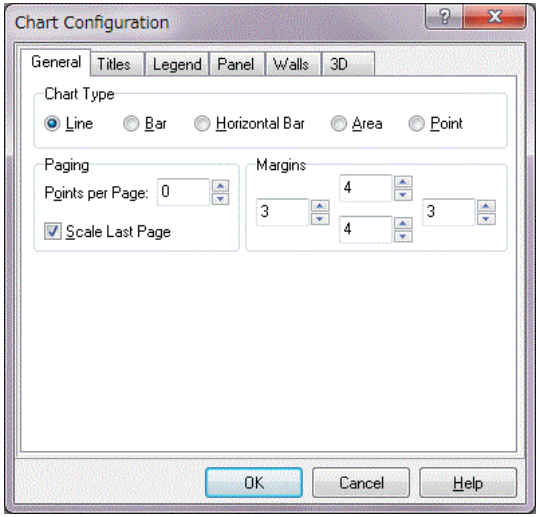
Data Analysis 画面



グラフ装飾
データコピー
印刷出力
グラフ出力
種類選択
コンフィギュレーション
終了

グラフ表示のオプション設定画面

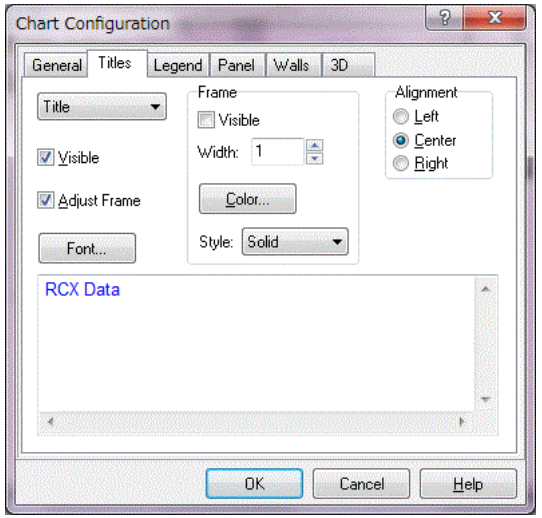
Chart Configuration/General画面



※Chart Configuration

グラフ表示のタイトル表示設定画面

Chart Configuration/Titles 画面



ツール群ウィンドウ（５）

グラフ表示選択（フォーマット）による出力（クリップボード、ファイル出力の選択も可能）

Chart Configuration/Legend 画面

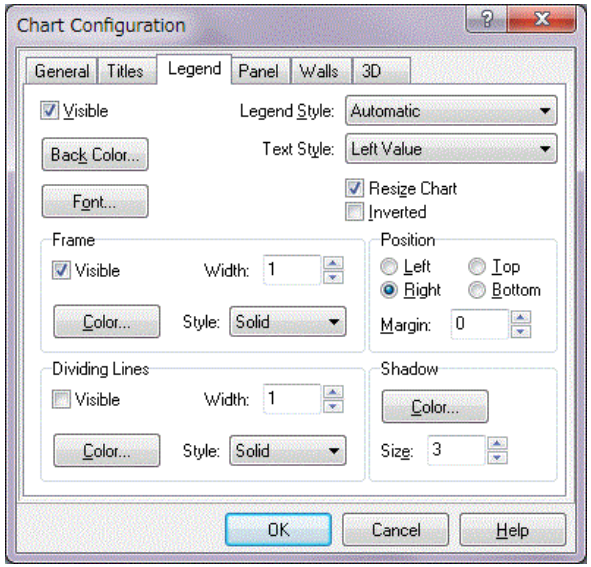
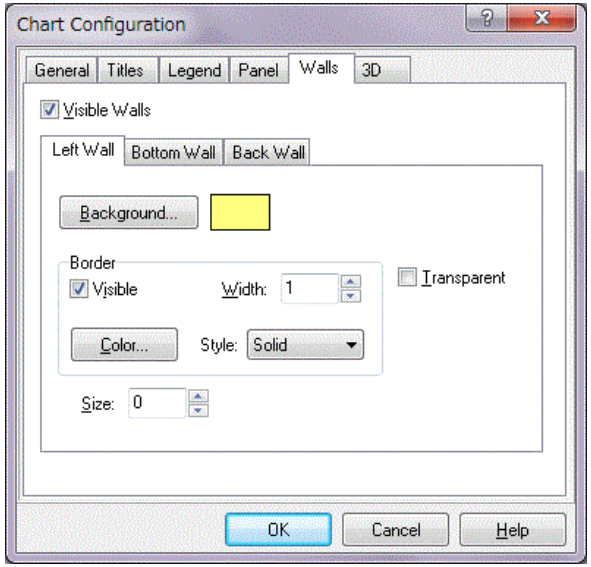


Chart Configuration/Walls 画面



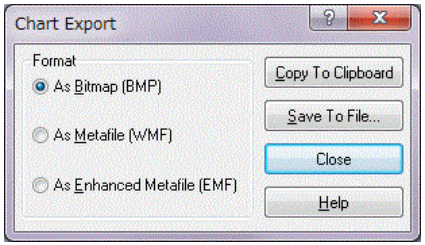
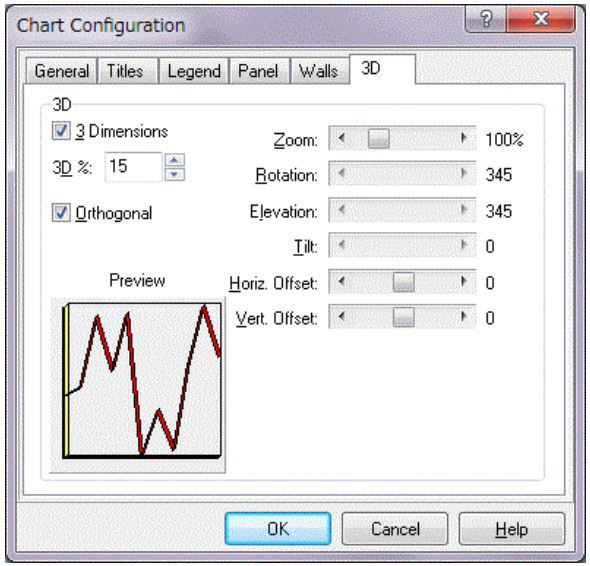


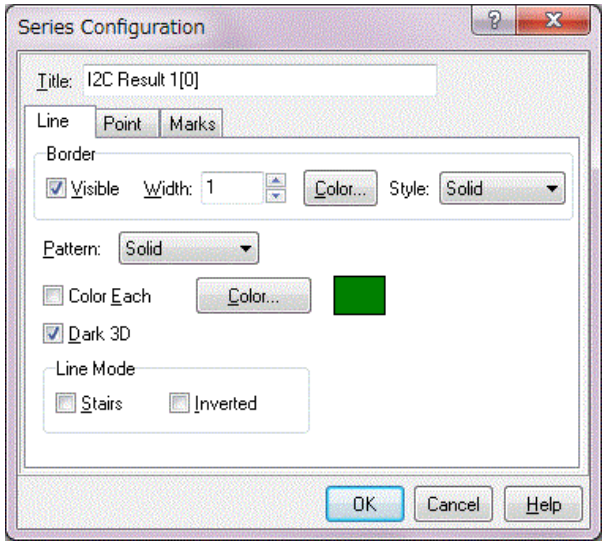
Chart Configuration/３D 画面



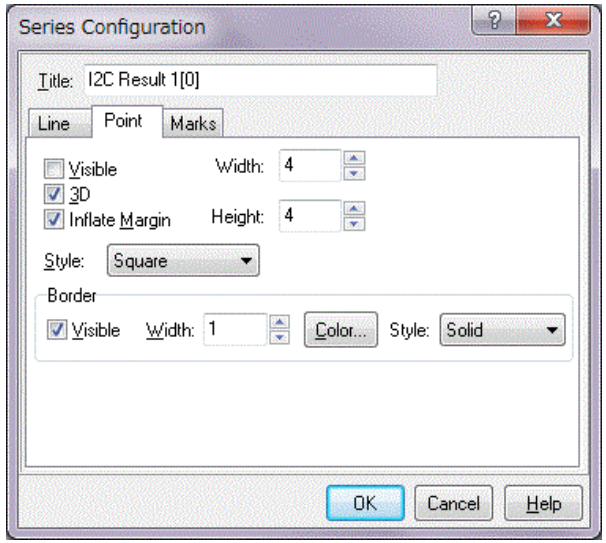
ツール群ウィンドウ（６）

I2Cセンサーの環境設定

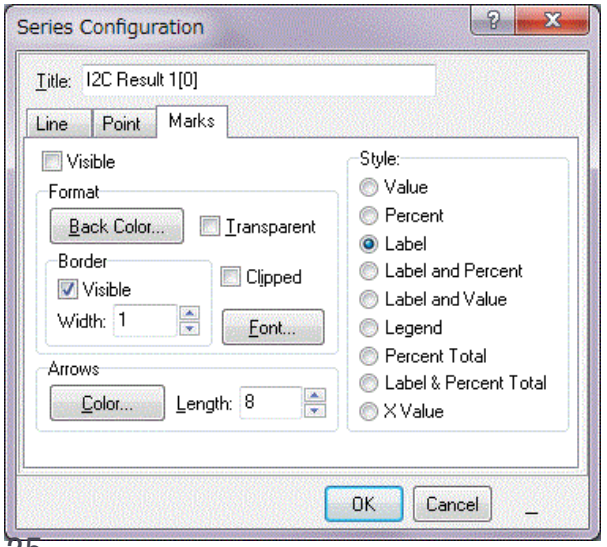
Series Configuration/Line 画面



Series Configuration/Point 画面



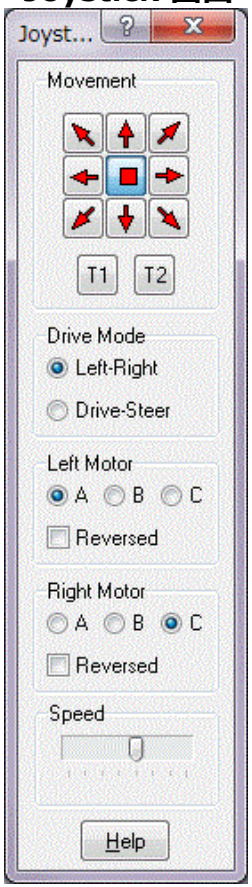
Series Configuration/Marks 画面



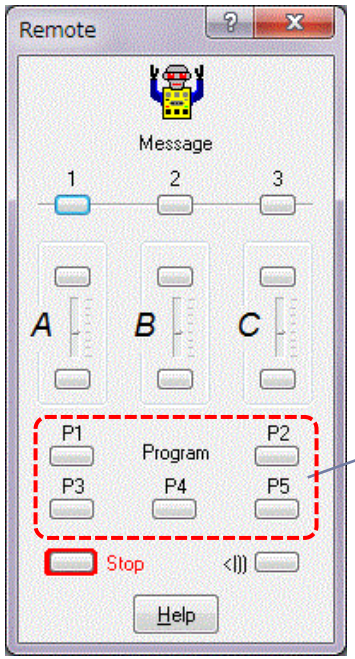
ツール群ウィンドウ（７）

ここでは、ダイレクトにNXTを起動する機能群を紹介。１つはジョイスティックによるもので、もう１つはリモート操作となる。

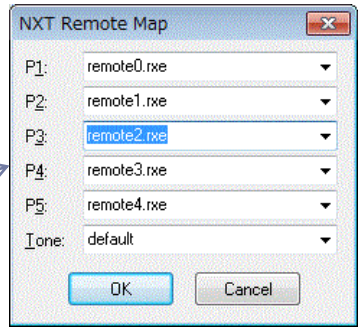
Joystick 画面



Remote 画面



Remote/NXT Remote Map画面

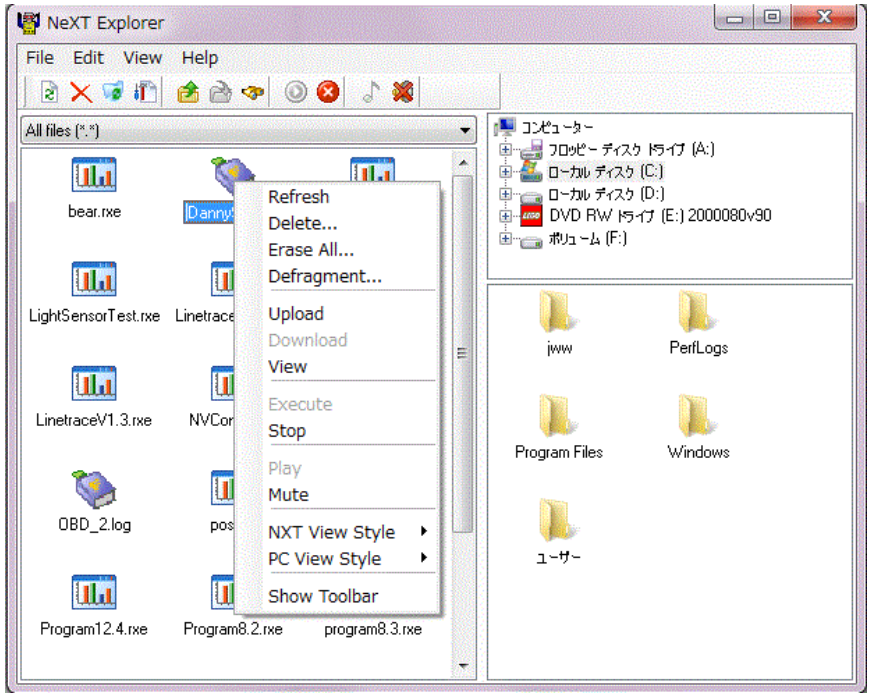


プログラムP1からP5までに、あらかじめ既存プログラムを組み込んでおくことで、任意にプログラムをこのP1からP5のボタンで実行できる。

ツール群ウィンドウ（８）

NXT上のフラッシュメモリーとPC側のNXT関連（NXCほか）
ファイルのエクスプローラで、メモリー上の確認なども可能。

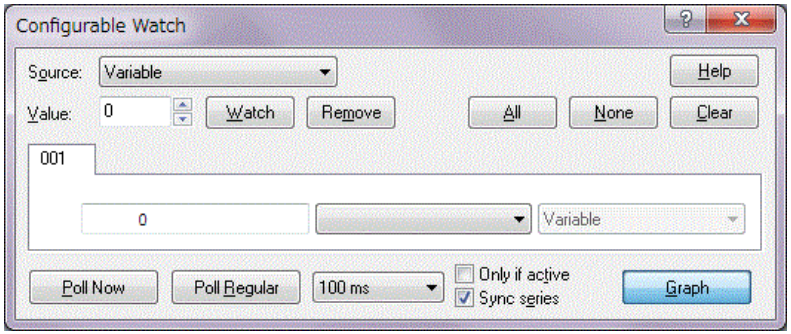
NXT エクスプローラ画面



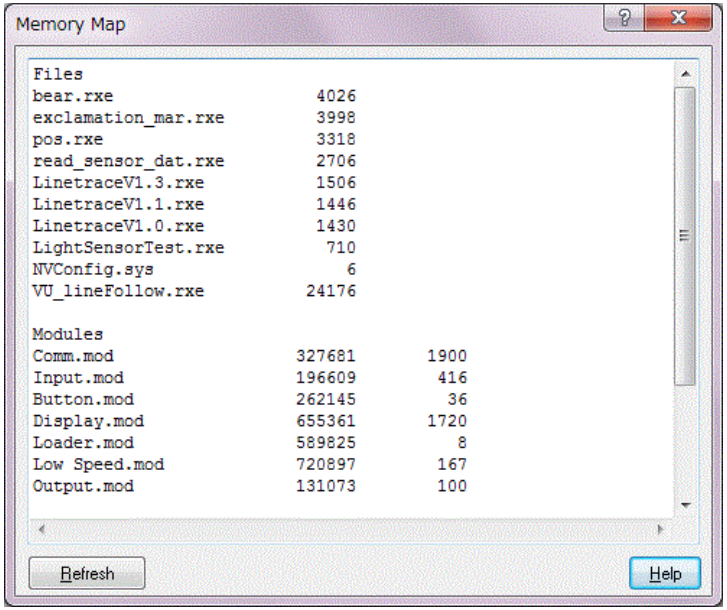
ツール群ウィンドウ（９）

調整設定を行う

Configurable Watch画面



NXTメモリーマップ画面

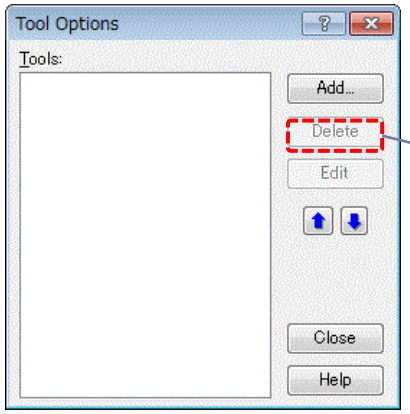


Messages画面

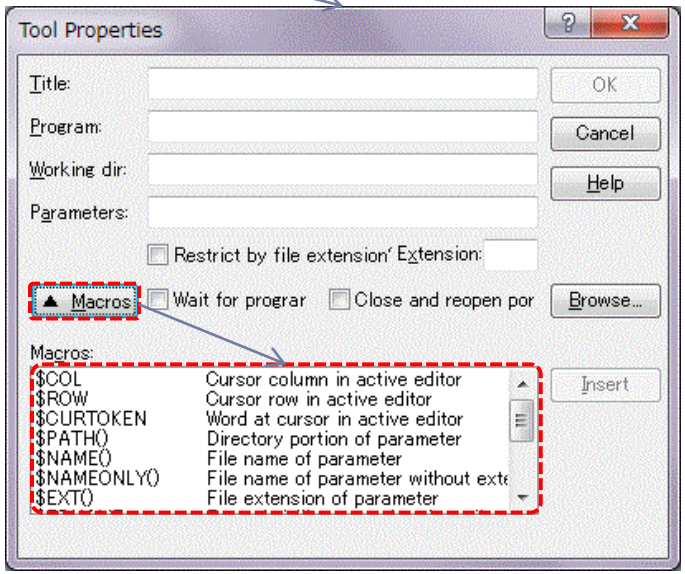
ツール群ウィンドウ（１０）

ツールオプション関連を設定

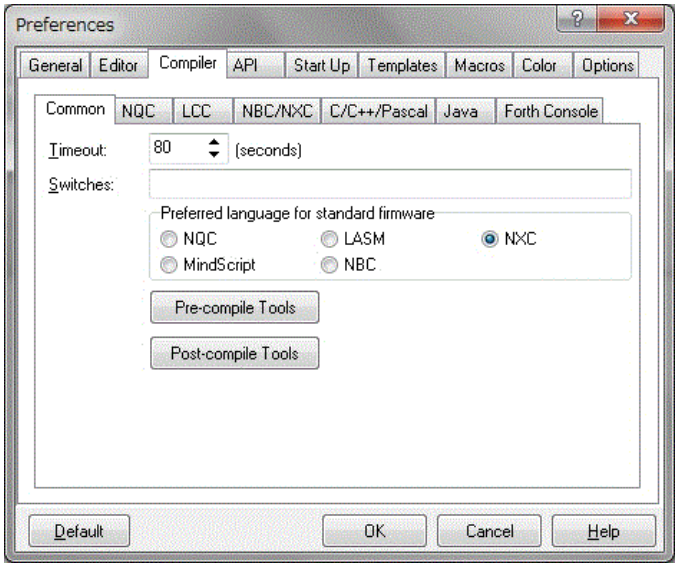
Tool Option画面



Tool Properties画面



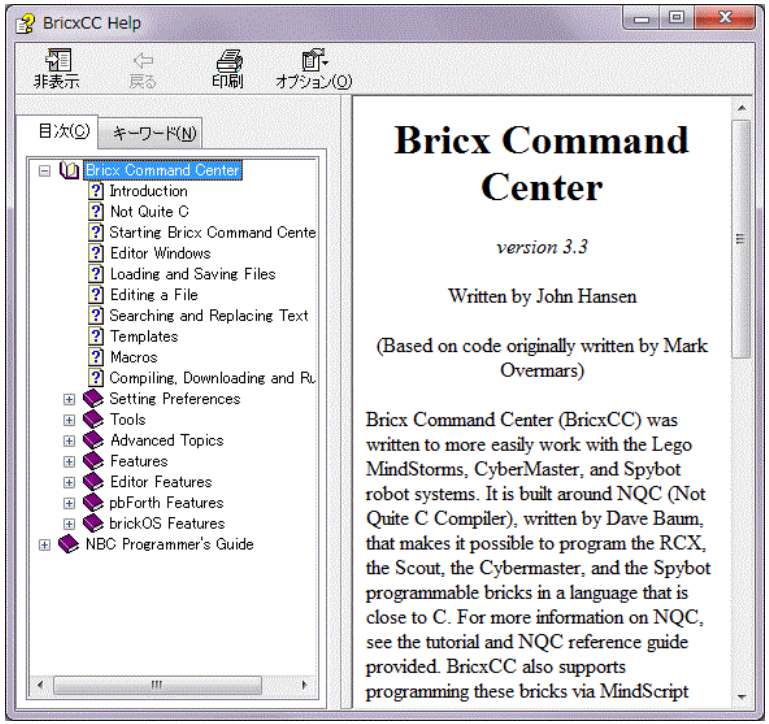
Preferences/Compiler/Common画面



ヘルプ表示

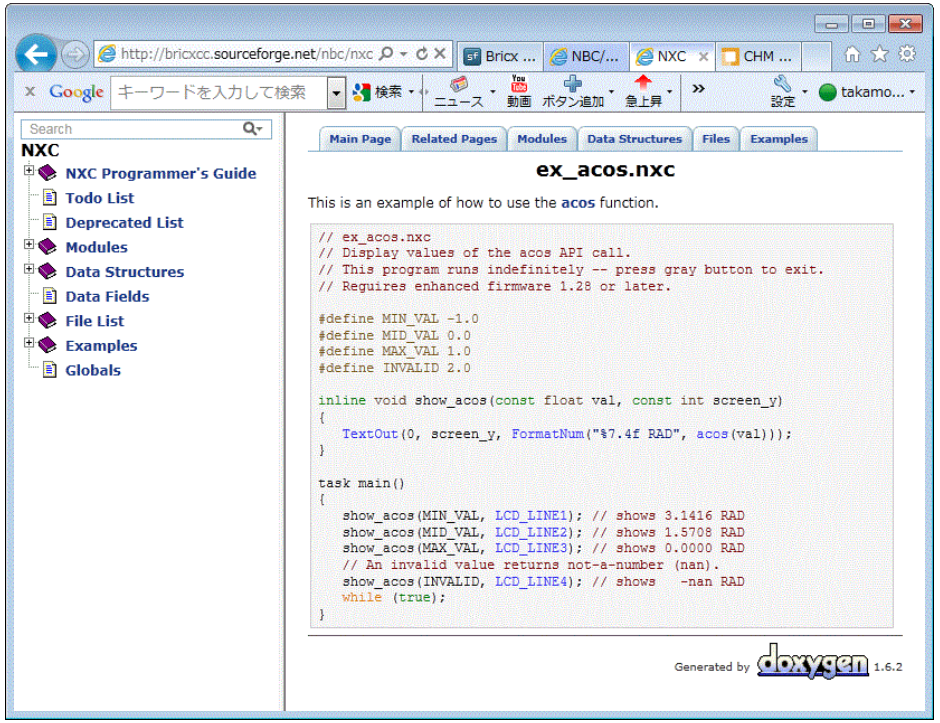
「Help」メニューから、BricxCC関連のヘルプを表示させることができる。

BricxCCのヘルプ画面



【注意】BricxCCを立上げ時点で、何もNXCプログラムを読み込んでいない状態でHELP画面を表示させるときに出てくる。
一度、NXCプログラムを読み込んで、HELP画面を表示させると、NXCのヘルプ画面に切り替わる。

NXC関連のヘルプ画面



【注意】NXCの関数や宣言関連など、詳細な情報が表示される。また、事例プログラムの豊富に組み込まれている。互いにリンクもされていて、操作が簡単。

Ⅲ.　NXCプログラミング

チュートリアル・ガイド

1．はじめてのプログラム作成

このチュートリアルでは、初歩的なNXC言語の紹介と最低限の「**BricxCC**」の使い方を紹介。

ここでは、NXC言語のいくつかの重要事項を学ぶ。

① まず互いのプログラムには、名前「**main**」を持つタスクが一つ必要であり、それによってロボットが実行できる。

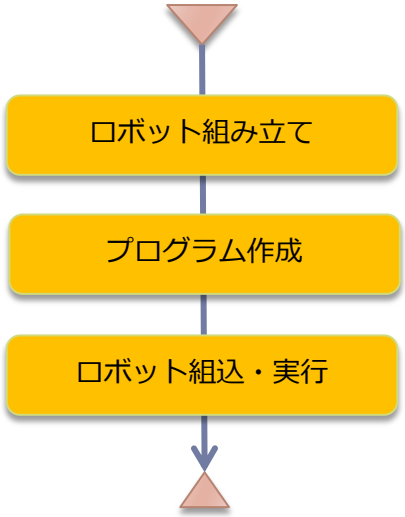
② それに、４つのモータに関する基本的なコマンド（**OnFwd()**、**OnRev()** および**Off(**」）と継続を意味する **Wait()** について学ぶ。

覚える事

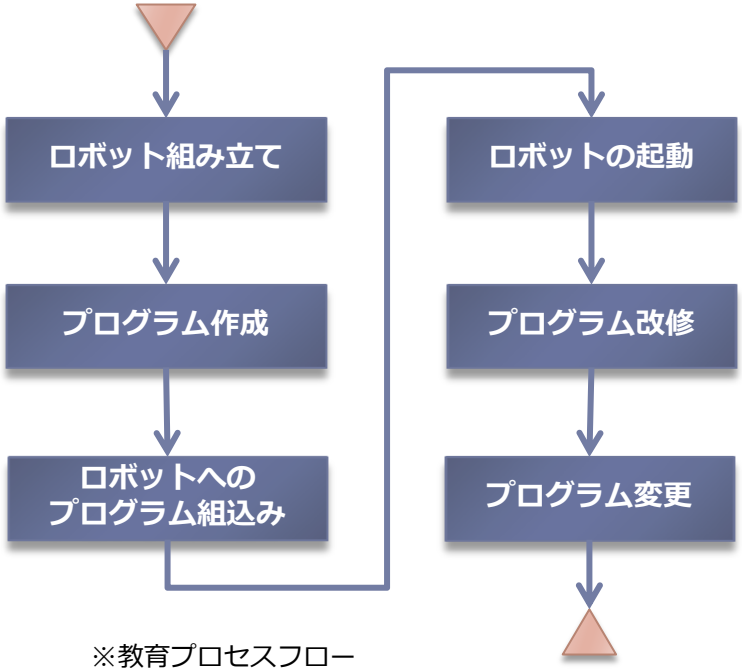
1. ロボットの組み立て
2. 「Bricx Command Center」の起動
3. プログラムの作成
4. プログラムの実行
5. プログラム内のエラー（誤り）
6. スピードの変更

モータ	標準出力ポート
右車輪	A
その他	B
左車輪	C

※この教材で推奨するモータとポートの関係



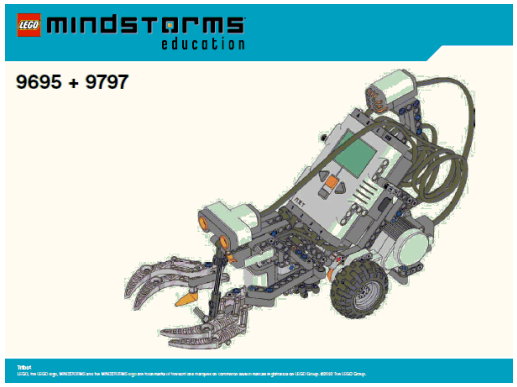
※ロボット作成・実行フロー

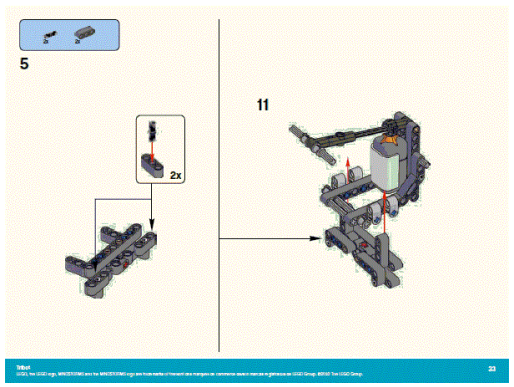


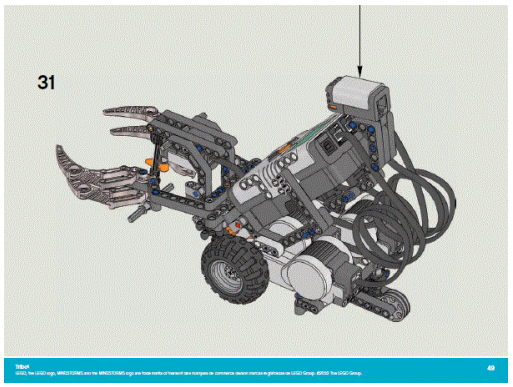
※教育プロセスフロー

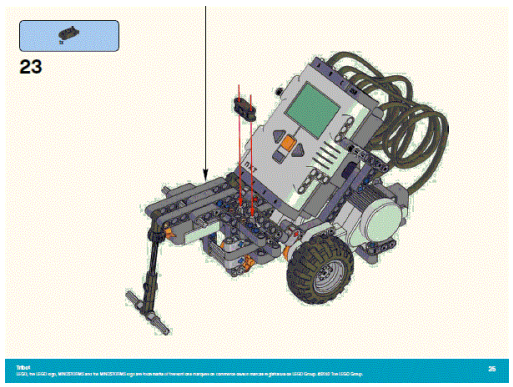
ロボットの組み立て

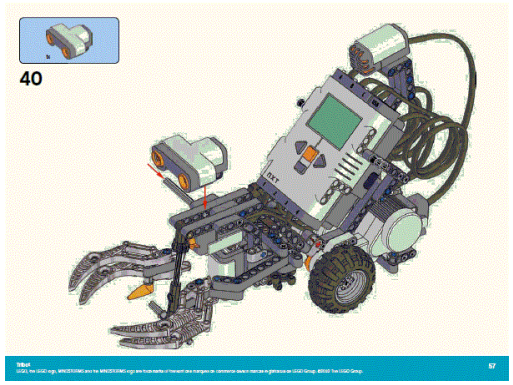
本教育カリキュラムで使う「**Tribot**」ロボットを手順図に従って組立て

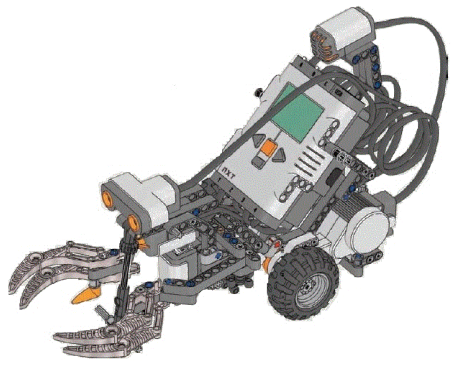








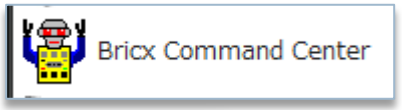




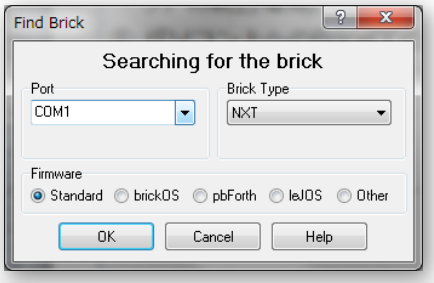
※Tribotは、２つのモータと１つの車輪で動き、各種センサーを取り付けてできている単純ロボット。
手順図に沿って簡単にロボットを組み立てることができる

「Bricx Command Center」の起動

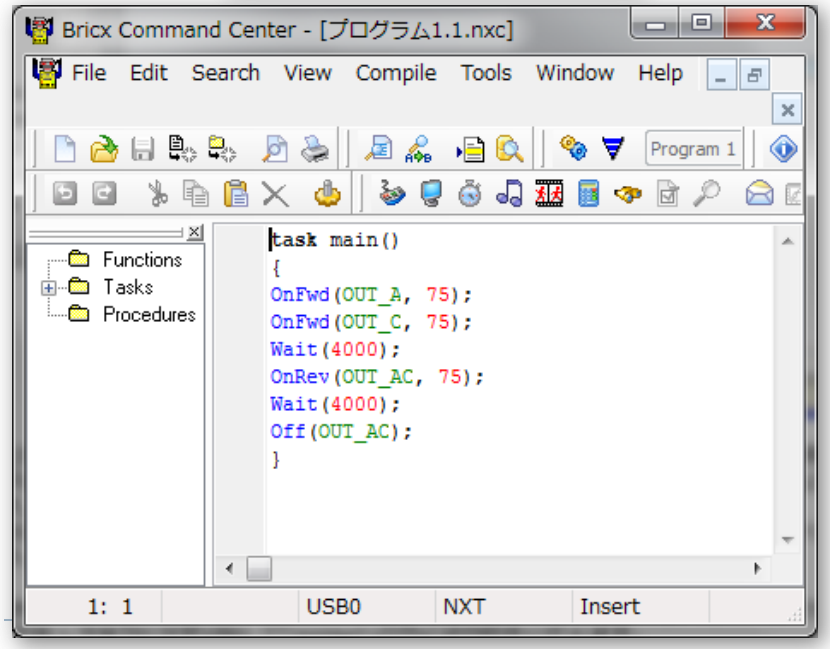
- BricxCCは、レゴ・マインドストームを動かす開発言語NXC（C言語ライク）の統合環境でフリーソフト。
  - NXC(**Not eXactly C**)は、NXTへの組込みプログラム言語
  - BricxCC（**Bricx Command Center**）は、NXCの開発統合環境を提供
  - 既に多くの情報や実績がネットや出版物で紹介

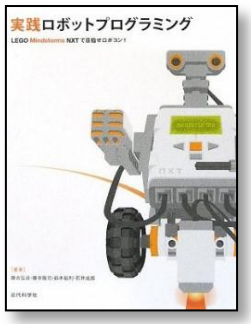


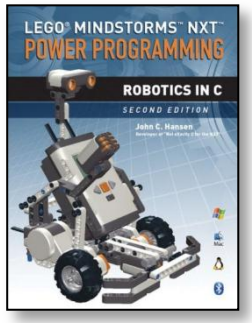
① パソコン上へのBricxCCのダウンロード
  - http://sourceforge.net/projects/bricxcc/files/bricxcc/



② BricxCCの環境設定（初期に行うべき設定）
  - 初期環境設定
  - 日本語コメント対応
  - 行番号表示設定
  - 色表示設定
  - コンパイル環境など







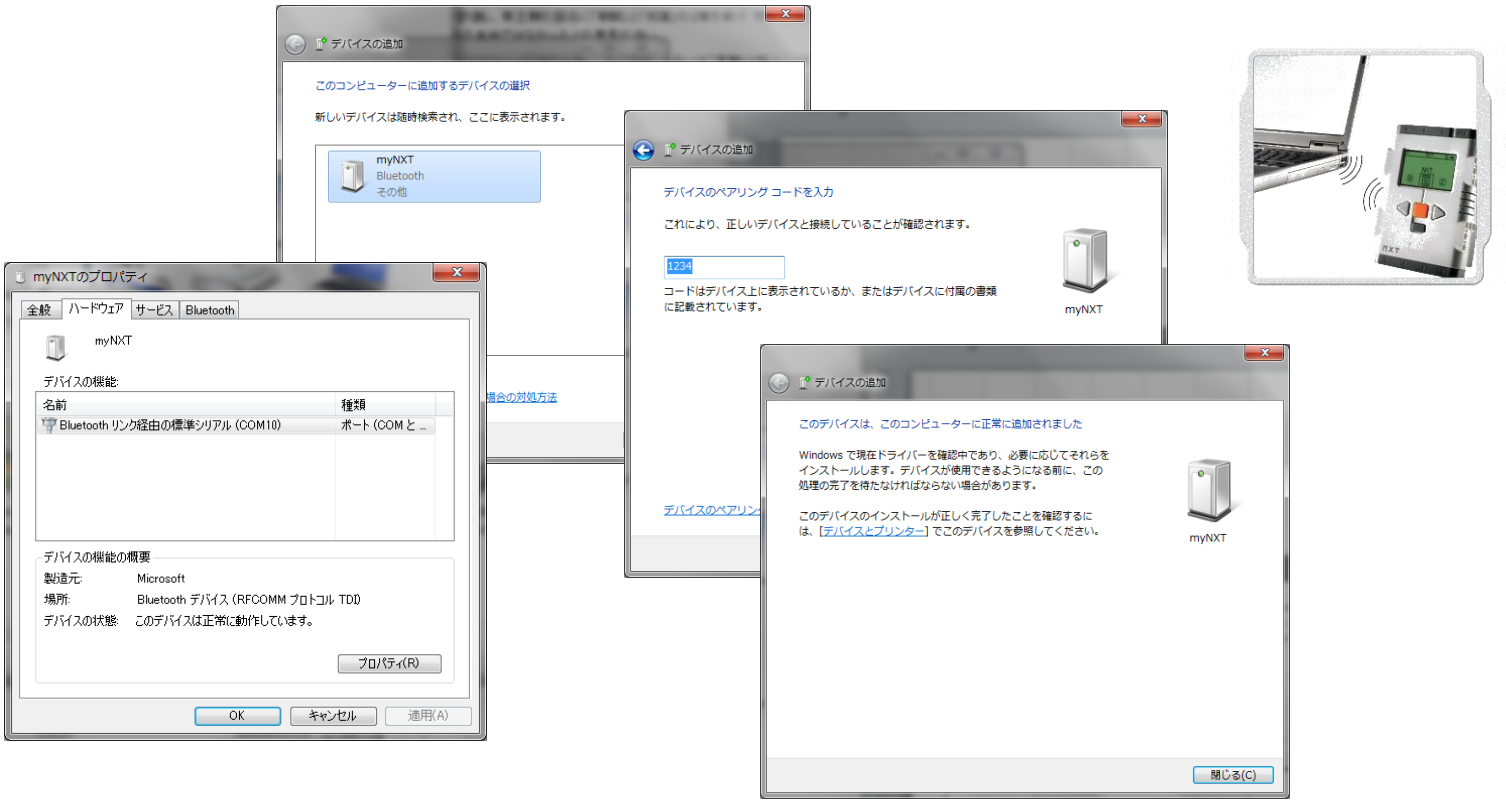
※BricxCC（NXC）関連本

※BricxCC（NXC）画面

TABrain

※Bluetoothの接続について

- 作成したプログラムをNXTへダウンロードし実行する際、USB接続（有線ケーブル）とBluetoothで対応。
- さらに複数NXT間どうしの通信をBluetoothで対応（NXTは最大4台まで）

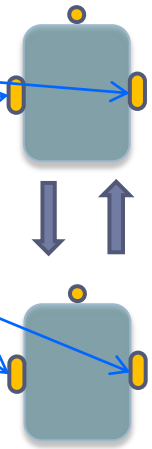


プログラムの作成

簡単なプログラムを作成し、NXTへダウンロードして、実行してみよう。

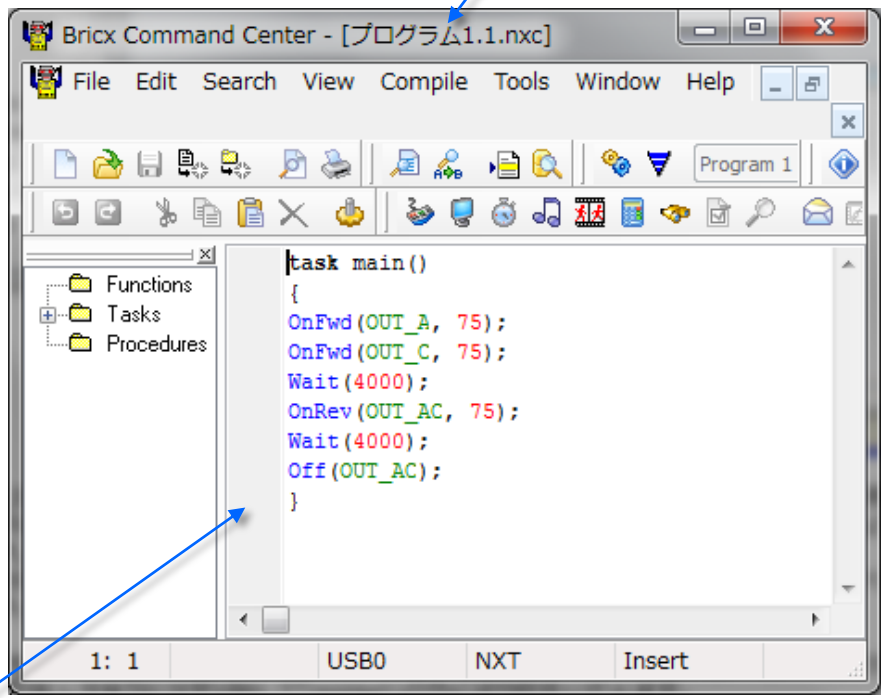
① 簡単なプログラムをBricxCC上のエディタで作成 （【File】→【new】でエディタに）
② プログラムの作成（ロボットを４秒間前進させ、その後４秒間後退し、停止する）
③ コンパイル & ダウンロード（【Compile]】→【Compile]】および【Download】実施）
④ プログラムの実行（【Compile】→ 【Run】）

```
task main()
{
  OnFwd(OUT_A, 75);
  OnFwd(OUT_C, 75);
  Wait(4000);
  OnRev(OUT_AC, 75);
  Wait(4000);
  Off(OUT_AC);
}
```



簡単なプログラム事例
（4秒前進、4秒後退、後停止）

※ファイル名を付けた上で、コンパイルすること
ファイル名が無いとコンパイラーエラーとなる。



BircxCC上でのプログラム作成エディタ

複数のプログラムソースを編集可能

※ プログラムソースのチェック

「**task**」（タスク：仕事）は、 コマンド（命令）とよび、中括弧内に、ひとつひとつのステートメントを定義していく。
「**task** main()」は、メインプログラム名で、最初に実行するプログラムセグメント。必ず必要となる関数定義となる。

「OnFwd」 は、 モータを前進させる関数。
引数の「OUT_A」および「OUT_C」は、**出力ポート名**となる。「75」は、モータの**パワー**となる。

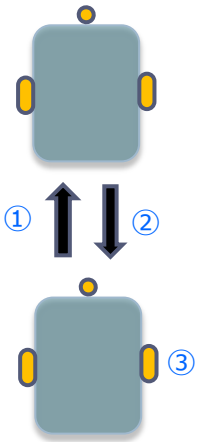
「Wait」 は、待ち（待機）時間を設定する関数。
引数の「4000」は、ミリ秒で、4秒間を意味する。

「OnRev」 は、モータを後退させる関数。
引数の「OUT_AC」は、出力ポートの「A」と「C」の同時設定となる。

「Off」 は、モータを停止する関数。ここでは、「OUT_AC」で「A」と「C」の両方を停止設定となる。

```
task main()
{
  OnFwd(OUT_A, 75);
  OnFwd(OUT_C, 75);
  Wait(4000);
  OnRev(OUT_AC, 75);
  Wait(4000);
  Off(OUT_AC);
}
```

①4秒前進
②4秒後退
③停止



※ C言語は、一般的なプログラム言語

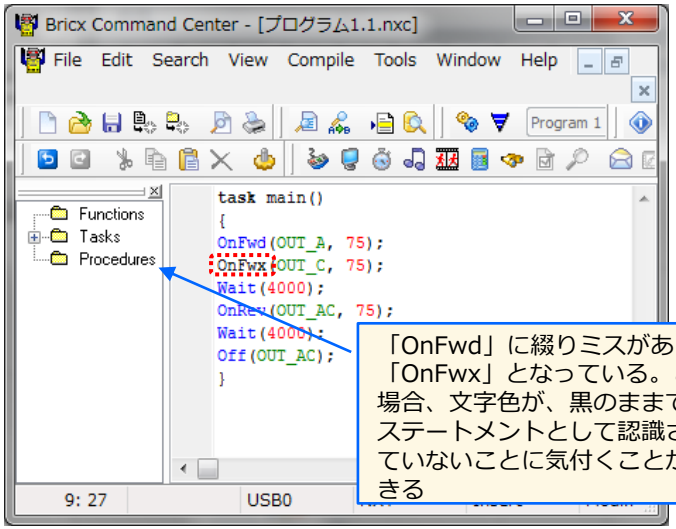
プログラムは、コマンドと呼ばれる
ステートメントの組み合わせで構成される。

```
task main()
{
  statements1;     ← ステートメント1
  statements2;     ← ステートメント2
 …
}
```

【覚えること】
OnFwd（出力ポート, パワー）
OnRev(出力ポート, パワー)
Off(出力ポート)
Wait(待機時間)
task main（） { 処理群 }

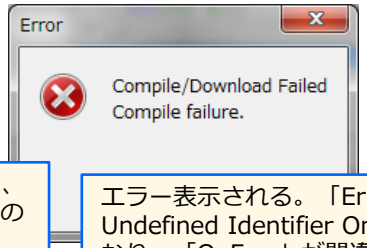
プログラム内のエラー【誤り】

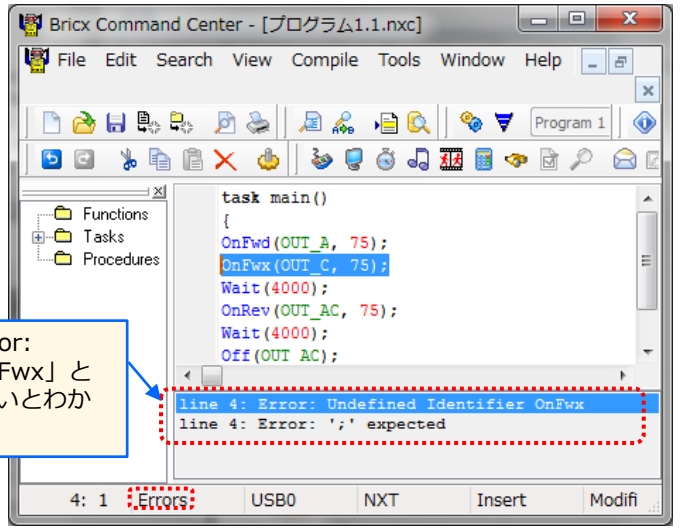
プログラムにある文法エラーやアルゴリズムエラーなどを、修正し、正しく動くようにする行為を**デバック**と呼ぶ。



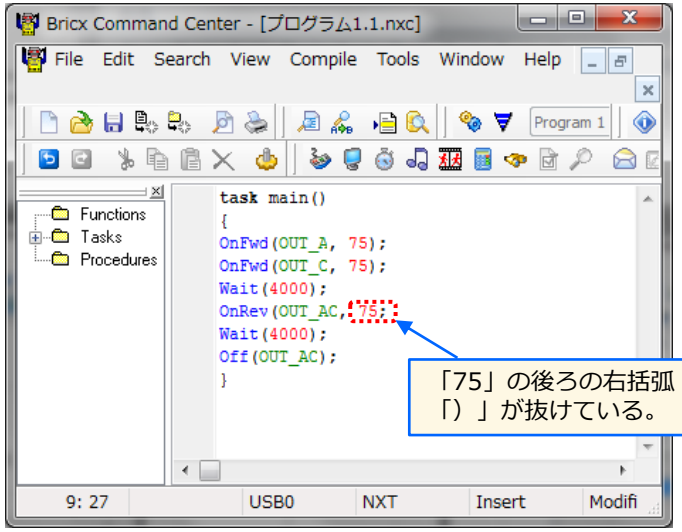
「OnFwd」に綴りミスがあり、「OnFwx」となっている。この場合、文字色が、黒のままで、ステートメントとして認識されていないことに気付くことができる

コンパイル
エラー表示



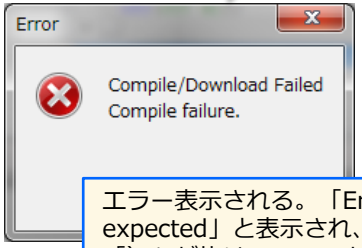


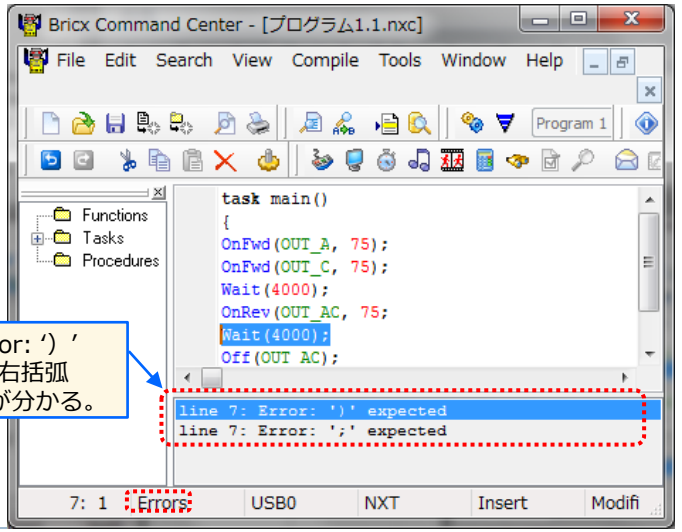
エラー表示される。「Error: Undefined Identifier OnFwx」となり、「OnFwx」が間違いとわかる。



「75」の後ろの右括弧「）」が抜けている。

コンパイル
エラー表示





エラー表示される。「Error: ')' expected」と表示され、右括弧「）」が抜けていることが分かる。

【注意】ファイル名を付けて保存していない場合、コンパイルするとエラーとなる

TABrain

スピードの変更【プログラム改善・修正】

移動するスピードを「ゆっくり」に変更するには

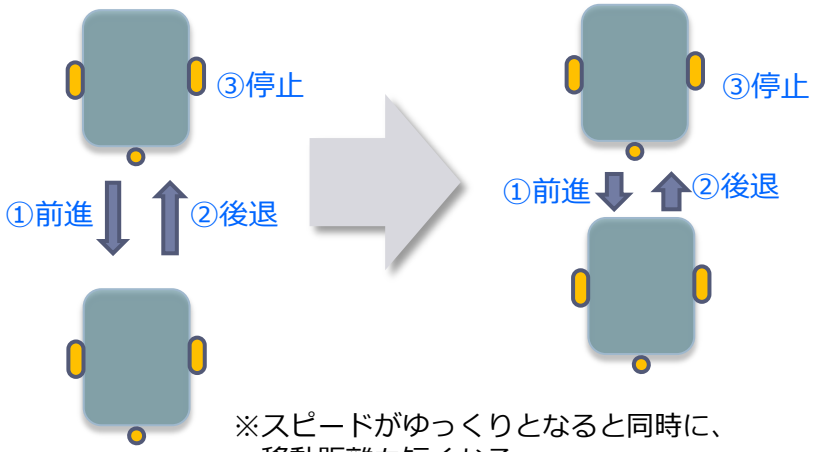
ロボットのスピードは早いか？　スピードを変更する場合には、コマンド内（OnFwdやOnRevなど）の２番目のパラメータ（引数）を変更。**モータのパワー**は、「０」から「１００」までの数値で設定。「100」 の場合が最大スピードで、「０」は停止（Stop）の意味。

```
task main()
{
  OnFwd(OUT_A, 30);
  OnFwd(OUT_C, 30);
  Wait(4000);
  OnRev(OUT_AC, 30);
  Wait(4000);
  Off(OUT_AC);
}
```

モータAのスピードを75から30にスピードダウン

モータCのスピードも75から30にスピードダウン

モータAとモータCのスピードを同時に、75から30にスピードダウン（②後退）



※スピードがゆっくりとなると同時に、移動距離も短くなる

【展開】

- ここで紹介した関数群の使い方についてまとめておく。

関数群	概　　説
Off	**モータの即時停止（OFF）** void Off(outputs): byte outputs: 出力ポート OUT_A　(0x00) OUT_B　(0x01) OUT_C　(0x02) OUT_AB　(0x03) OUT_AC　(0x04) OUT_BC　(0x05) OUT_ABC (0x06)
OnFwd	**モータの前進** void OnFwd(outputs,pwr): byte outputs: char pwr:
OnRev	**モータの後退** void OnRev(outputs,pwr): byte outputs: char pwr:
Wait	**ミリ秒単位(1秒=1000ミリ秒)での待機時間の設定** void　Wait(ms) : 戻り値なし unsigned long ms : 待機する1ミリ秒単位の数値
Stop	**実行中のプログラムの停止** void Stop(bvalue) : 戻り値なし bool bvalue: ブーリアン値で真のときプログラム停止

- 先ずは、躊躇せずに、すぐにプログラムを作成し、コンパイル実行を学ぶ必要がある。

まとめ

- この章では、「BricxCC（Bricxコマンドセンター）」を使って、NXCの最初のプログラム作成について紹介。

- どうやってプログラムを作成するか、どうロボットにダウンロードするか、またどうプログラムを実行させてロボットを動かすかが分かった。

- この他にも「BricxCC」では、多くの機能を装備している。いろいろと触ってみたり、付属のドキュメント（英文）を読んだりして勉強してみよう。

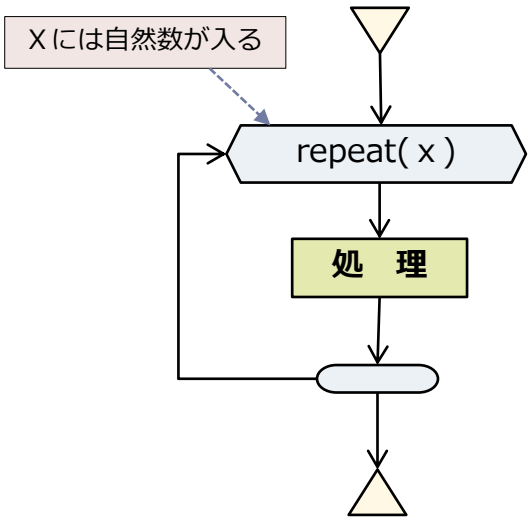
2．より興味のあるプログラム作成

最初に紹介したプログラムは、特に驚くものではなかった。もっと興味のあるプログラムに挑戦してみよう。ここでは、NXC言語プログラムについて、さらにいくつかの機能を段階的に紹介していく。

この章では、**「repeat」ステートメント**の使い方と、**コメント**について学ぶ。それに、カッコを使った**入れ子での関数の扱い**や、**インデントの使い方**も学ぶ。また、ここでは、ロボットをさまざまな経路で動かすことができるようになる。さらに、プログラムをいろいろと変更してみて、ロボットを動かしてみよう。

覚える事

1．繰り返し文　repeatステートメント
2．#define コマンド
3．コメントとは
4．入れ子（ネスト）の関数とは
5．インデント(段下げ)



※repeatのフローチャート図

旋回プログラム（１）

ロボットの片方か両方のモータを停止したり、逆回転したりすることで、ロボットの方向を変えてみることが可能。

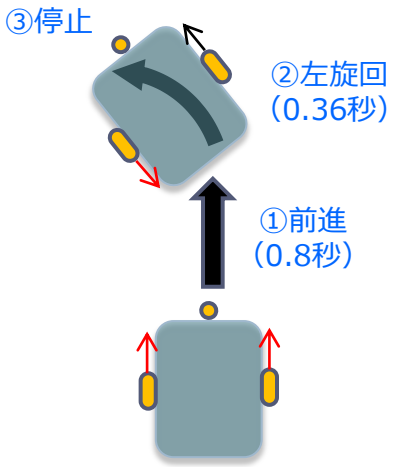
次のプログラムでのロボットは、少し前進し、右回転。

```
task main()
 {
     OnFwd(OUT_AC, 75);
     Wait(800);
     OnRev(OUT_C, 75);
     Wait(360);
     Off(OUT_AC);
}
```

①前進

②左旋回

③停止



※ロボットの実行

旋回プログラム（２）

この中の２つめの継続コマンド「**Wait()**」を360から500に変え、回転角度がどう変わるかを試してみる。ロボットがどう回転するかは、この数値によることが分かる。

```
task main()
 {
     OnFwd(OUT_AC, 75);
     Wait(800);
     OnRev(OUT_C, 75);
     Wait(500);
     Off(OUT_AC);
}
```

360から500に変更

この数値をプログラム上で、簡単に変更できるように、数値に名前を付けてみる。NXCでは、**定数定義「#define」**コマンドを利用。

```
#define MOVE_TIME 1000
#define TURN_TIME 360
task main()
 {
     OnFwd(OUT_AC, 75);
     Wait(MOVE_TIME);
     OnRev(OUT_C, 75);
     Wait(TURN_TIME);
     Off(OUT_AC);
 }
```

#define 定数定義

①前進時間

②旋回時間

最初の2 行で、定数二つを定義。ここで定義した定数は、プログラム内で使うことが可能。

この定数の定義は、２つのメリットがある。
① プログラムを読みやすくすること
② 簡単に変数を変えることができること

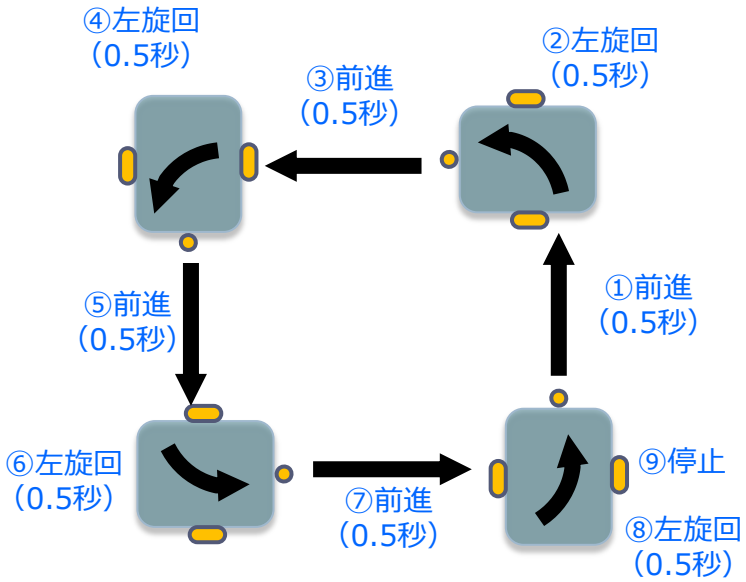
【注意点】「**BricxCC**」では、定数定義のステートメントの文字カラーを独自に設定することが可能。➡ ６章参照

コマンドの繰り返し（１）

つぎに、ロボットの動きとして正方形を描くようにしてみよう。正方形の意味とは、ロボットが前進し、９０度回転し、再度前進し、９０度回転しの繰り返しで、もとの位置に戻るようにすること。ここでは、４回の前進と90度回転の繰り返しのプログラムを、**「repeat」ステートメント**を使って簡単になるプログラミング記述を紹介。

この「repeat」ステートメントの中の数値（４）は、**繰り返しの回数**を意味し、さらに**中カッコ｛と｝とを使って**、その繰り返しの範囲を明確にする。

それから、上記のプログラムでは**段下げ（インデント）**を使っています。これは必ずしも必要ではありませんが、**プログラムを読みやすく**する上では重要。




```
#define MOVE_TIME 500
#define TURN_TIME 500
task main()
 {
--->repeat(4)
--->{
--->---> OnFwd(OUT_AC, 75);
--->---> Wait(MOVE_TIME);
--->---> OnRev(OUT_C, 75);
--->---> Wait(TURN_TIME);
--->}
--->Off(OUT_AC);
}
```

4回繰返し

前進

左旋回

停止

---> インデント

【注意事項】
必ずしも四角形を描く訳ではありません。床の状態やバッテリーの状態、タイヤの状態などで旋回時の回転は変わることで、調整が必要となります。

コマンドの繰り返し（２）

それでは、前頁のプログラムをさらに１０回の繰り返しを行ってみる。
このプログラムでは、「**repeat**」ステートメントの中にさらに「repeat」が使われています。この状態を「**入れ子（nested：ネスト）**」と呼びます。好きなように「repieat」ステートメントを使って入れ子を作ることができますが、開始「｛」と終了「｝」のカッコをペアで使い、さらに**段下げ（indent:インデント）**を使うことに心がけてください。

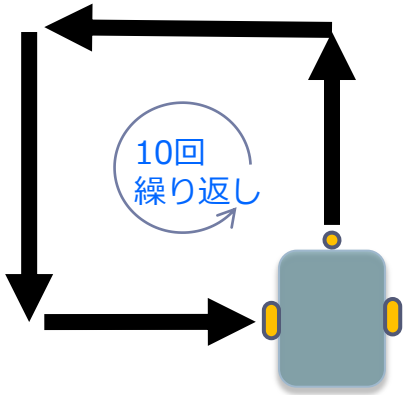
```
#define MOVE_TIME 1000
#define TURN_TIME 500
task main()
{
  repeat(10)
  {
    repeat(4)
    {
      OnFwd(OUT_AC, 75);
      Wait(MOVE_TIME);
      OnRev(OUT_C, 75);
      Wait(TURN_TIME);
    }
  }
  Off(OUT_AC);
}
```

入れ子（**ネスト**）

段下げ（**インデント**）

10回の繰り返し

正方形を描く繰り返し



【注意】必ずしもこのプログラムで、正確に正方形を描くような動きはしません。バッテリーの状態やタイヤの材質、床の材質などによって、動きが異なります。より正確に動くようにするには、TURN_TIMEの数値を調整してみる必要があります。この様な調整をすることがロボットのプログラミングでは重要となります。

コメントの追加

プログラムをより読みやすくするためには、コメントを入れる。各行の**先頭に「//」を入れる**と、その行の実行を無視し、コメントとして扱う。長いコメントの場合には、**「/*」と「*/」で囲んで**コメントとする。「BricxCC」では、**コメント文は、傾斜文字に変更**される。

「/*」 から 「*/」 まで がコメントペアとなっていることが条件

「//」以降から改行までがコメント

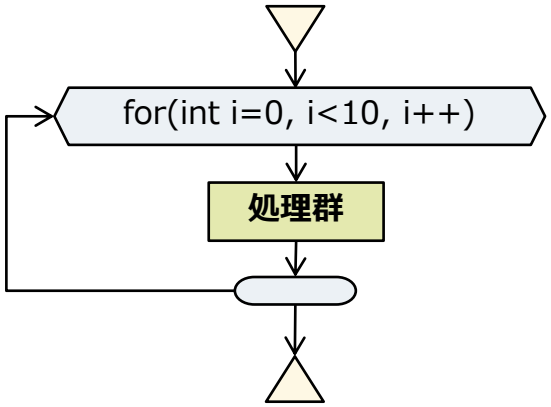
```
/* 10 SQUARES (10回の４角形)
   This program make the robot run 10 squares（本プログラムはロボットが１０回４角形を描く）
*/

#define MOVE_TIME 500 // Time for a straight move（前進する時間）
#define TURN_TIME 360 // Time for turning 90 degrees（回転する時間）
task main()
{
  repeat(10)              // Make 10 squares（１０回繰り返し）
  {
    repeat(4)
    {
      OnFwd(OUT_AC, 75);
      Wait(MOVE_TIME);
      OnRev(OUT_C, 75);
      Wait(TURN_TIME);
    }
  }
  Off(OUT_AC);            // Now turn the motors off
}
```

【展開】

- 繰返しプログラムの標準ステートメントは、以下の**for**文がある。

```
for （式　，繰返し条件，式2）　本体;
```

```
for （int i=0, i<10, i++）　{
  処理群
}
```



※for文のフローチャート図

- 効率良いプログラミングについて

項　目	概　　説
インデントを使う	括弧の対応などが分かり易くなる
関数・変数名を分かり易く	プログラムが読みやすくなる
コメントを入れる	分かり易くなり保守性が高まる
モジュール化を図る	分かり易いモジュール構成とする
処理は上から下へ	Goto文は極力使わない
ダイレクトな数値は避ける	意味を持つ定数名などを使う

- プログラミングの注意

NXCのコンパイラは、上から下へコンパイルしていく

このことで、関数名が定義されずに、宣言されて利用しようとした場合には、エラーとなる。

関数やタスクが他の関数やタスクを読込もうとした際には、それより前に、関数やタスクの宣言が必要となる。

タスクの事前宣言は、以下のとおり

task name**();**

関数の事前宣言は、以下のとおり

return_type name**(**argument_list**);**

型、関数名、引数群ともに定義文と同じ記述とする。

まとめ

- この章では、プログラムの繰り返しのための「**repeat**」ステートメントの使い方を学んだ。
- また、マクロの定義として、**#define**コマンドを学んだ。
- それに、プログラムを見やすくするために、**コメント**を入れることや、カッコを使った**入れ子（ネスト）**での関数の扱い、それに**段下げ（インデント）**の使い方も学んだ。
- ここまでで、ロボットを**さまざまな経路で動かす**ことができるようになった。
- つぎの章に進む前に、プログラムをいろいろと変更してみて、ロボットを動かしてみてみよう。

３．変数の使い方

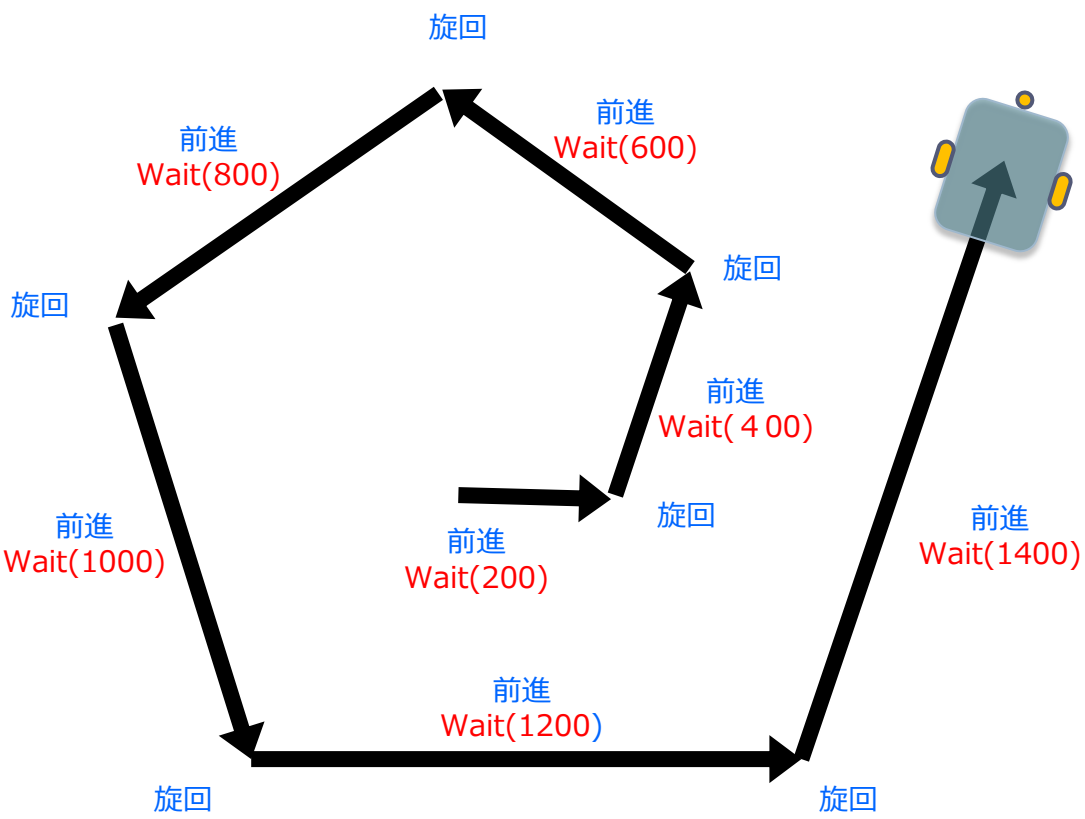
変数は、どのプログラミング言語でも非常に重要なもの。変数は、**値をメモリに保存**し、**いろいろなステートメントで利用**でき、**値そのものを変更**することもできる。ここでは、いろいろな事例を使って変数の使い方を覚えよう。

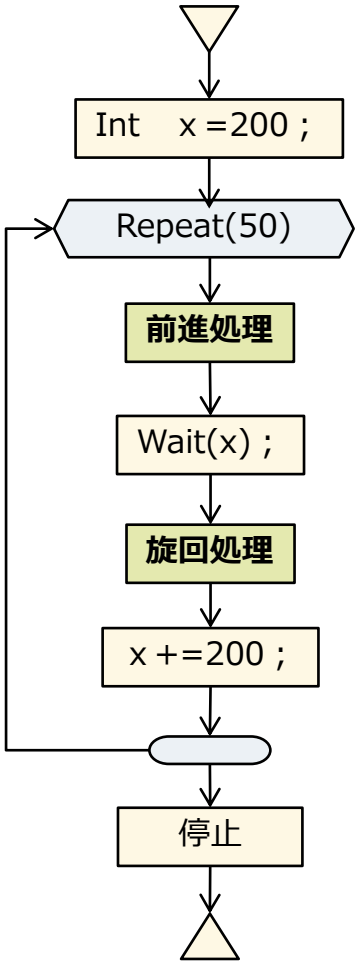
覚える事

１．定数と変数
２．変数定義、値設定、変数利用
３．配列定義、配列初期値設定
４．算術代入演算子（+=、-=、*=、/=）
５．乱数（ランダム数）発生
６．whileステートメント

渦まき（螺旋）起動（１）

前章で紹介したプログラムを変更してロボットを**渦まき状に動かす**。
本プログラムは、つぎつぎに前進する継続時間を増やすことで実現できる。



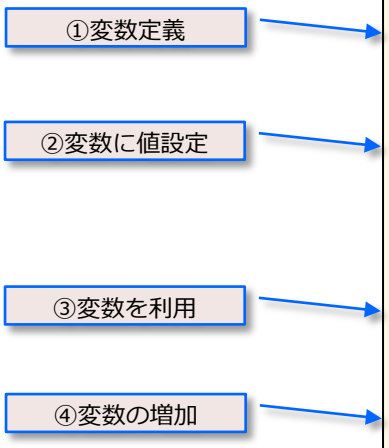


渦まき起動の
プログラム・フローチャート

渦まき（螺旋）起動（２）

この場合、前述において時間設定の「MOVE_TIME」は**定数**で、値を変化させることはできない。値を変化させるには、**変数**を使う必要がある。変数は、NXCではつぎのように簡単に定義できる。

以下にプログラムを紹介する。




```
#define TURN_TIME 360
int move_time;            // define a variable (変数定義)
task main()
{
  move_time = 200;     // set the initial value (変数の初期化)
  repeat(50)
  {
    OnFwd(OUT_AC, 75);
    Wait(move_time);   // use the variable for sleeping(継続時間の変数使用)
    OnRev(OUT_C, 75);
    Wait(TURN_TIME);
    move_time += 200; // increase the variable (変数値の増加)
  }
  Off(OUT_AC);
}
```

ここで注意すべき行にコメントを付加している。

①変数定義： キーワード「**int**」ステートメントを使って、任意の名前をつけた変数を定義。（一般に、小文字を使った名前を変数名として、大文字を使った名前を定数名として使う）この変数名は、英文字で始まり、中に英数字やアンダーラインの文字を含むことができる。（同様に、定数名およびタスク名も同じことが言える）この「**int**」は、整数の変数であることを意味し、整数のみがここで定義される。

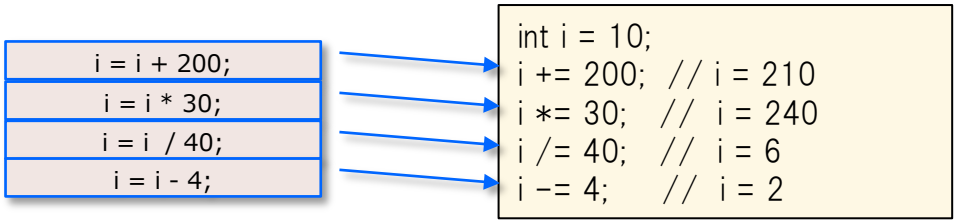
渦まき（螺旋）起動（３）

②変数に値設定：　つぎのコメント行では、この変数「move_taime」に200を設定している。これ以降の行で、この変数を200として利用することができる。

③変数を利用：　つぎの「repeat」ステートメント内のループ（繰り返し）においては、継続時間にこの変数「move_time」を使っている。

④変数の増加：　さらにループの終わりには変数値を200増加（プラス）させるプログラムとなっている。ループ内の初回は、継続時間は200ms（ミリ秒）で、つぎの回は400msで、さらに３回目は600msで繰り返す。

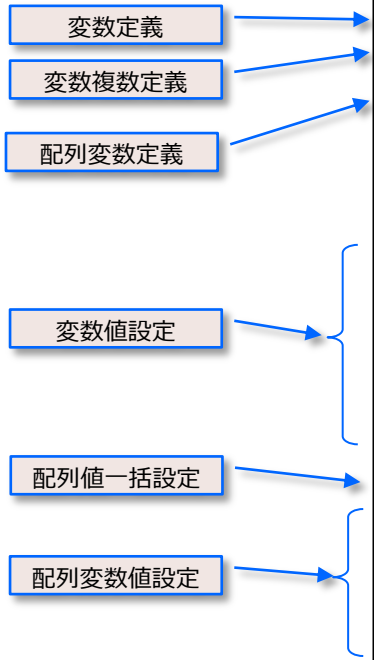
【参考】　**算術代入演算子：** このループの最後の行にある「**+=**」は、左の変数名「move_time」に入っている数値に、右の数値200を増加（プラス）することを意味する。同様に「***=**」は、左の変数名に入っている数値に、右の数値を掛け合わせる意味があり、「**-=**」は右の数値で引き算し、「**/=**」は右の数値で割り算することを意味する。（注意：割り算の場合には、丸め処理した整数が算出されます）さらに変数を他の変数に加算したり、より複雑な式に使ったりすることできる。



渦まき（螺旋）起動（４）

つぎの例では、**変数の使い方の例を紹介**。最初の2行では、編集を複数定義できることを紹介。この２行で定義した３つの**変数**（aaa,bbb,ccc）を１つの行で定義することもできる。

つぎの行の変数名「values」は**配列**定義で、複数の数値を扱うことができ、この配列は鍵カッコ「[]」内のインデックス（参照値）を使って利用。




```
int aaa;
int bbb, ccc;
int values [] ;

task main()
{
  aaa = 10;
  bbb = 20 * 5;
  ccc = bbb;
  ccc /= aaa;
  ccc -= 5;
  aaa = 10 * (ccc + 3);       // aaa is now equal to 80 (aaa の値は 80)
  ArrayInit(values, 0 , 10);  // allocate 10 elements =0 (配列10 個の要素を0 に設定)
  values[0] = aaa;
  values[1] = bbb;
  values[2] = aaa*bbb;
  values[3] = ccc;
}
```

NXCでの**整数配列**は、つぎの様に定義。
int name[]**;**
以下の関数は、変数名「values」に10個の**配列要素**を用意し、値をすべて０（ゼロ）に設定するもの。
ArrayInit(values, 0, 10);

乱数【ランダム数】（１）

ロボットを動かすとき、ランダムな数値を使って、予想外の動きをするようしてみよう。

どう動くかランダムな数値は、興味ある値となる。

　＜毎回数値が変わることを**ランダム**と呼ぶ＞

NXCでは、**乱数（random number）**を発生する関数「**Randam（整数）**」があり、この乱数を使ってロボットを動かすよことができる。

ここでのロボットは、ランダムな時間で前進し、ランダムな角度で回転（左旋回）する。

このプログラムでは、２つの変数を定義し、それらにはランダム数値を設定している。

「**Random(600)**」は、乱数関数で、リターン値として０から600までの数値が返ってくる。（最大値600は、リターン値には含まれない）

また変数を使わずに、直接「**Wait(Random(600))**」とすることもできる。

```
int move_time, turn_time;

task main()
{
  while(true)
  {
    move_time = Random(600);
    turn_time = Random(400);
    OnFwd(OUT_AC, 75);
    Wait(move_time);
    OnRev(OUT_A, 75);
    Wait(turn_time);
  }
}
```

乱数発生

前進

旋回

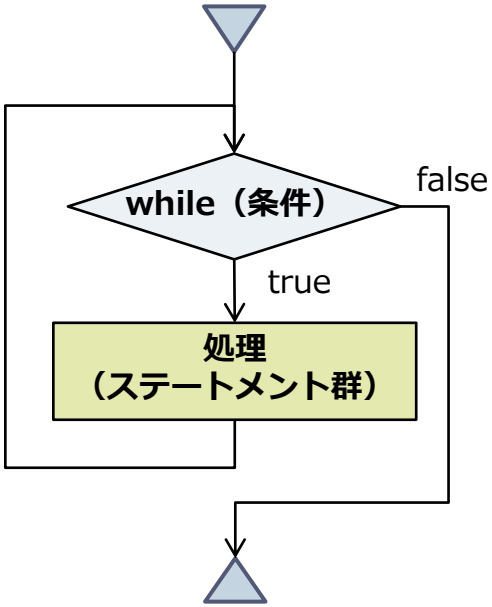
※ 乱数発生は、サイコロを振って出てくる目のようなもの

乱数【ランダム数】（２）

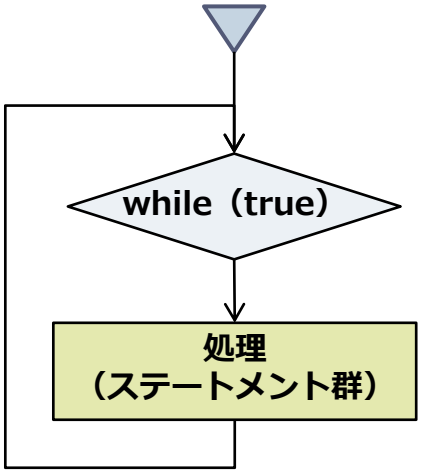
ここでは、新しいループとして、「**repeat**」の代わりに「**while(true)**」を使っている。
この「**while**」ステートメント（文）では、カッコ内がture（真）の状態の場合に、以下のカッコ内のステートメントが繰り返し実行される。

「while」ステートメント中のカッコ内の「**ture**」は特別な意味があり、常に真として、以下のカッコ内のステートメントを無限に繰り返し実行する。（ NXT上のグレーボタンを押すまでは繰り返す）

【参考】「**while**」ステートメントについては、さらに４章で学ぶ。



while文のフローチャート表現



※while文の条件がtrueの場合
無限ループとなる

while文による無限ループ
（繰り返し）

【発見】　true やfalseの代わりに、「１」や「０」を使うこともできる。
while(true)は、while(1)としてもコーディングできる。

【展開】

- 複数の値を持つ構造体の定義設定：**struct**文の宣言構文、および定義事例と利用例を紹介しておく。

```
struct  構造体名  {
      タイプ   変数名;
      タイプ   変数名;
         :
}
```

struct文の宣言構文

```
struct  Sperson  {
   int  age;
   string name;
   long ID;
}
```

struct文による定義事例

```
Sperson sman;
   :
sman.age = 40;
sman.name = “Ichiro”;
sman.ID = 9233412234;
```

構造体の利用例

- 配列 の高次配列や初期設定などは、以下のように行う。

```
int  One_array[];       //   一次配列
int  Twe_array[][];     //   二次配列
```

```
int  One_array[] = { 1, 2, 3, 4 };                              //   一次配列
int  Twe_array[][]={ { 2, 3, 1, 3 } { 1, 3, 2, 3 }} ;     //   二次配列
string  name[] = { “Taka”, “Mina”, “Sho”, “Masa”, “Kaai”}   //   文字配列
```

- **ArrayInit**文による配列の初期設定は、次のように行う。

関数名	機能/書式・型・パラメータ
ArrayInit	**配列の初期化** void ArrayInit(aout,value, count): 戻り値なし variant & aout[] : 初期化する対象の配列 variant value: 初期値 unsigned int count: 配列のうちの作成要素数

まとめ

- この章では、**変数**と**配列**の使い方を学んだ。
- NXCの主なデータタイプは以下のとおり。

型	ビット数	範囲・意味
byte, unsigned char	8	1バイト・1文字（正の整数値）
char	8	１文字(-128～127)
int	16	-32,768～32,767
short	16	-32,768～32,767
unsigned int	16	符号なし16ビット整数（0～65,535）
long	32	-2,147,483,648～2,147,483,647
unsigned long	32	符号なし整数（０～4,294,967,295）
float	32	実数値

型	意味
string	文字列
mutex	排他処理の変数
struct	**構造体**
Array[]	配列※
bool	true または false の値を表す

※配列宣言は、その他の型と配列型を宣言
　int Iarray[];
　string Sarray[][];
など

- またここでは、**乱数発生**について学び、ロボットを予想外の動きをさせることができた。
- 最後に、「**while**」ステートメントを使うことで無限に繰り返す方法を学んだ。

4．制御構造

これまでの章で、「repeat」および「while」ステートメントを学んだ。これらのステートメントは、プログラム内のほかの実行ステートメントを制御（コントロール）するもの。

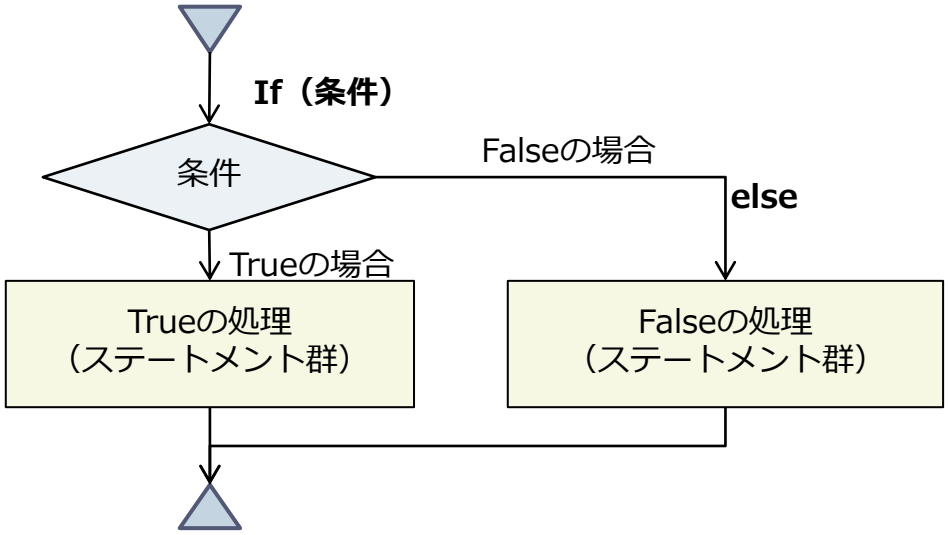
これらは、「**制御構造（control structures）**」と呼ぶ。この章では、以下の制御構造を学ぶ。

1） if　ステートメント

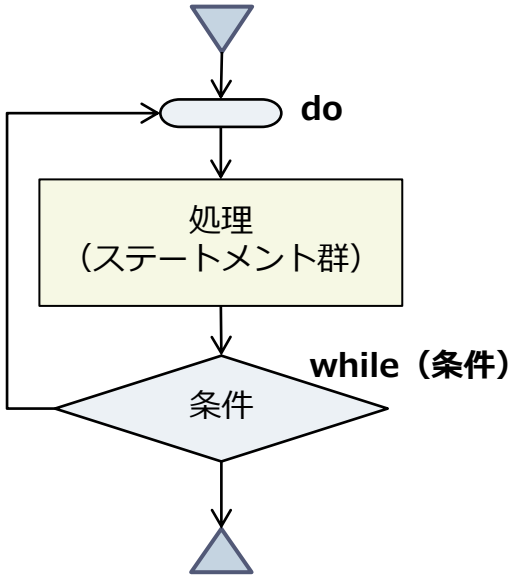
2） do　ステートメント

覚える事

1．制御構造
2．ifステートメント
3．doステートメント
4．関係演算子
5．乱数（ランダム数）発生
6．whileステートメント



If文のフローチャート表現



do文のフローチャート表現

if ステートメント（１）

if （条件）｛trueの処理ステートメント群｝ else ｛falseの処理ステートメント群｝

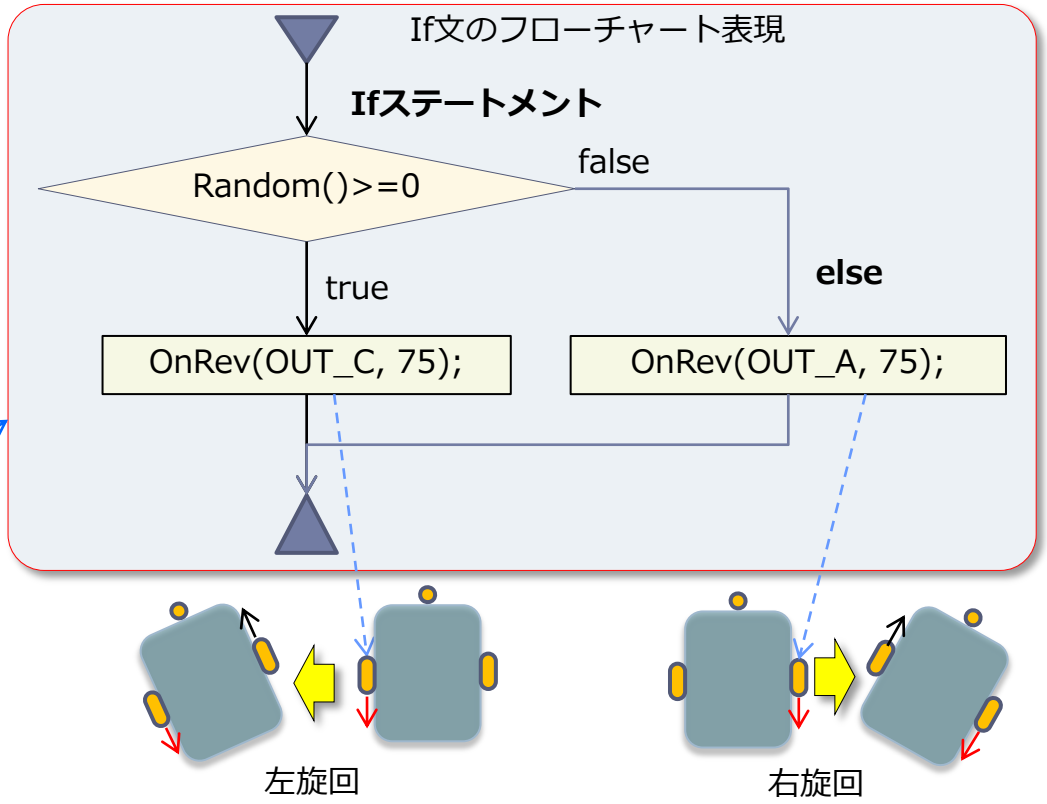
プログラムの一部分をある条件下で実行したいとき、「**if**」 ステートメントを使う。

事例は、これまで使ってきたプログラムを再び変更。ロボットを新たに旋回して動かす。

ロボットを前進させ、左右の何れかに旋回させる。　動作は再度乱数を使って実行させる。ゼロ（＝０）か正数（＞０）か、それ以外のゼロ未満の整数（＜０）かによって、右旋回させたり、左旋回させたりする。

以下にプログラムを紹介。

```
#define MOVE_TIME 500
#define TURN_TIME 360
task main()
{
  while(true)
  {
    OnFwd(OUT_AC, 75);
    Wait(MOVE_TIME);
    if (Random() >= 0)
    {
      OnRev(OUT_C, 75);
    }
    else
    {
      OnRev(OUT_A, 75);
    }
    Wait(TURN_TIME);
  }
}
```



if ステートメント（2）

「**if**」ステートメントの条件のカッコで囲まれた内容判定の状態が「true（真）」の状態のとき、つぎのカギ括弧内のステートメント群を実行。もし「false（偽）」であれば、「**else**」以下の鍵カッコのステートメント群を実行。

ここで使っているカッコ内の判定条件は

「Random() >= 0」

これは、乱数発生した数字が、ゼロ以上の整数の場合に、「**if**」のステートメントが「true（真）」となる。

ここで比較する関係演算子としてどのようなものがあるかを紹介する。

関係演算子	
==	等しい
＜	より小さい
＜=	以下
＞	より大きい
＞=	以上
！=	等しくない

true	常に真　（= 1）
false	常に偽　（= 0）
ttt != 3	ttt が3でない場合に真
(ttt >= 5) && (ttt <= 10)	ttt が5 以上、かつ10 以下の場合に真
(aaa == 10) || (bbb == 10)	aaa が10、あるいはbbbb が10 のとき真

また条件を組み合わせて使うこともでき、「and」の場合には「&&」を使い、「or」の場合には「||」を使う。

if ステートメント（3）

「**if**」ステートメントは、２つの**分岐処理**となる。

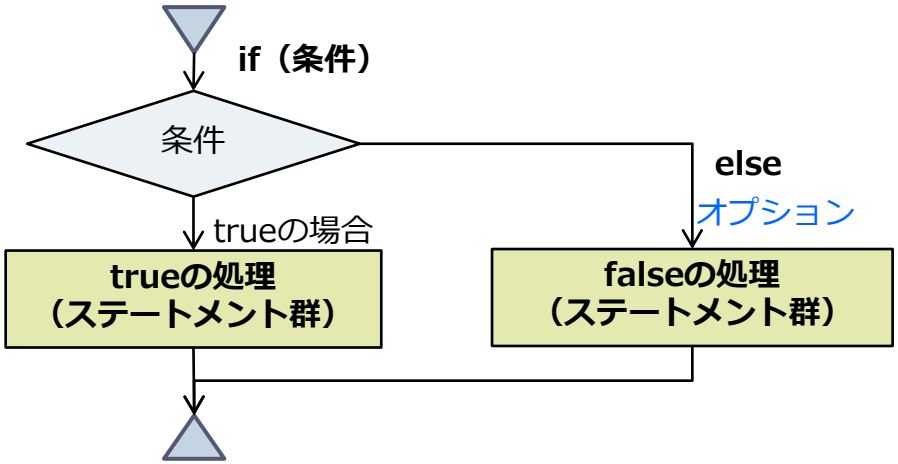
すぐつぎに続く中カッコ「｛と｝」内のステートメント群は、条件が真（true)の場合に実行。

偽(false)の場合には、その後に続く「**else**」ステートメント以下の中カッコ内のステートメント群が実行される。

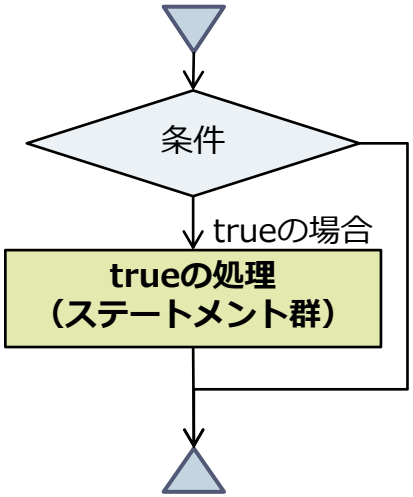
また、「**else**」以下の中カッコのステートメント群はオプションで不要な場合もある。

```
if (条件)
{
  trueの処理;
}
else
{
  falseの処理;
}
```

オプション



If文のフローチャート表現



※elseオプションが無い場合の処理

do ステートメント（１）

ほかの制御構造として、「**do**」ステートメントがある。

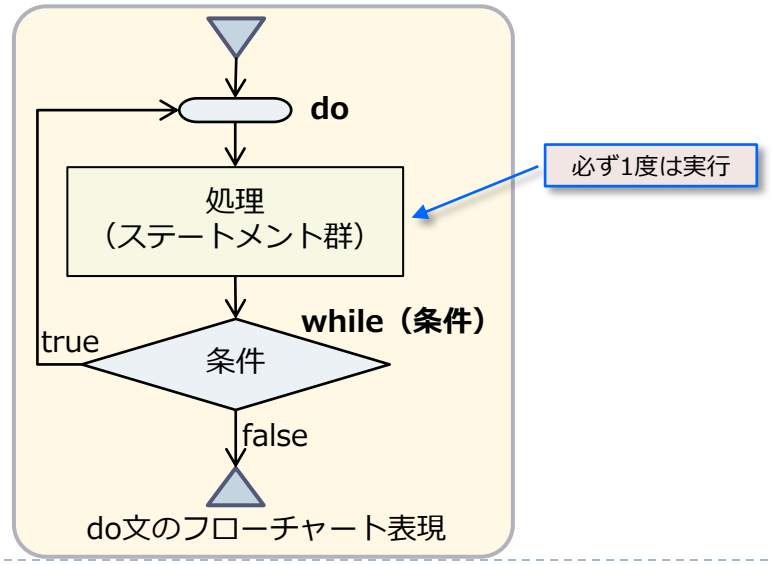
```
do
{
  ステートメント群;
}  while (条件);
```

必ず1度は実行

条件が真のとき
ステートメント
群を実行

「**do**」ステートメントは、「**while**」ステートメントと同様な状況で使用される。

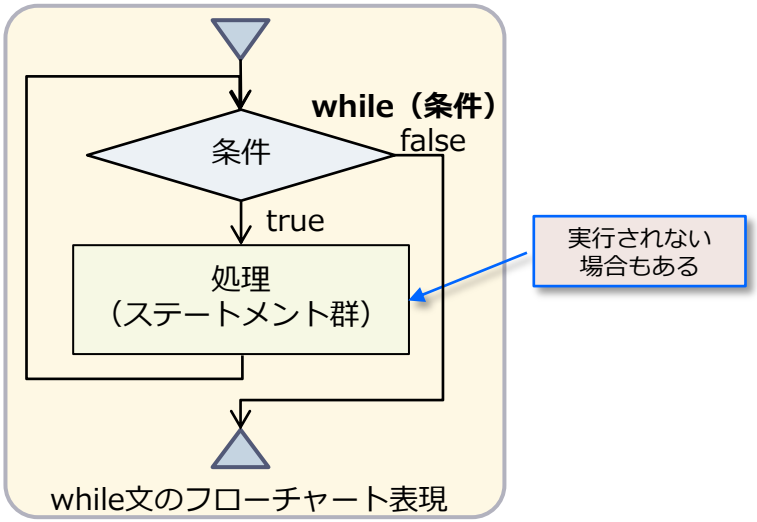
「**do**」ステートメントは、最初にステートメント群を実行した後、条件を評価（テスト）して繰り返し実行するかどうかとなる。つまり「**do**」ステートメントでは、少なくとも１回は、カッコ内のステートメント群を実行します。



「**do**」ステートメントは、つぎに続く中カッコを実行し、その後の「**while（条件）**」ステートメントの条件 が真(true)の場合には、さらにステートメント群を繰り返し実行する。

条件が偽(false)の場合には、繰り返し実行を行いません。この状態は「**if**」ステートメントを使っても表現できます。それでは、つぎの事例プログラムを使って、ロボットを20秒間、ランダムに動かし、停止（ストップ）させてみよう。

「**while**」ステートメントは、最初に条件を評価（テスト）し、以下のステートメント群を実行（繰り返す）するかどうかとなる。



do ステートメント（2）

```
int move_time, turn_time, total_time;
task main()
{
  total_time = 0;
  do
  {
    move_time = Random(1000);
    turn_time = Random(1000);
    OnFwd(OUT_AC, 75);
    Wait(move_time);
    OnRev(OUT_C, 75);
    Wait(turn_time);
    total_time += move_time;
    total_time += turn_time;
  }
  while (total_time < 20000);
  Off(OUT_AC);
}
```

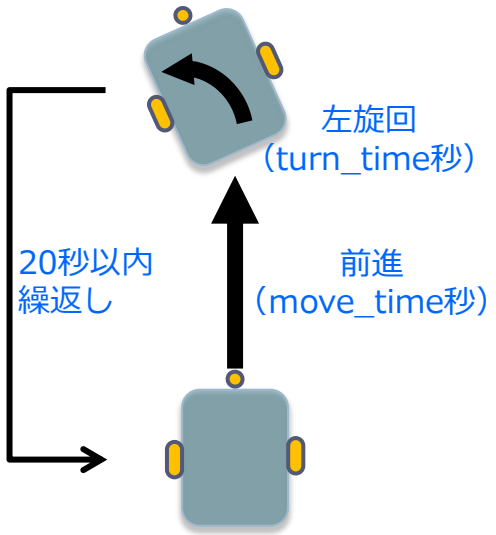
do文範囲

乱数発生

前進

左旋回

総合時間計算



上述の「**do**」ステートメントでは、まず乱数発生によって、前進する時間と、旋回する時間を設定している。

つぎに、前進し、左旋回している。

さらに、総合時間（total_time）を計算している。

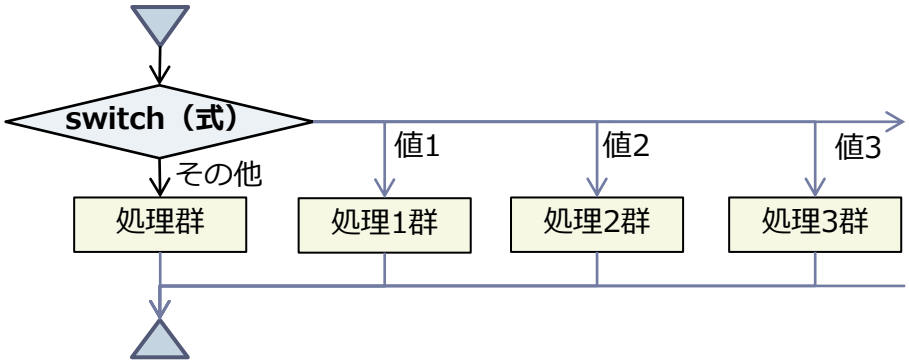
「**while**」において、総合時間がまだ20秒以内であれば、「do」ステートメントを繰り返している。

【展開１】

- 分岐処理では **i f** 文の他に、**switch**文がある。

式の値によって、それぞれの処理を行うことができる。

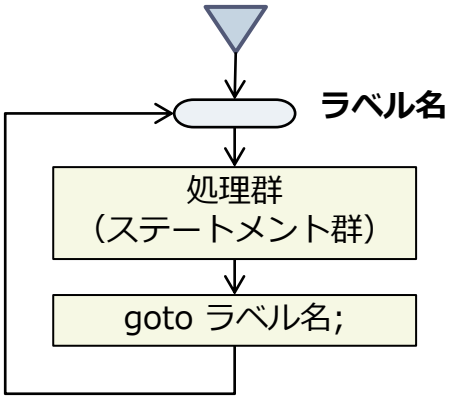
```
switch(式)
{
   case 値1:
       <処理１群>
       break;
   case 値2:
       <処理2群>
       break;
   case 値3:
       <処理3群>
       break;
       :
   default:
       <処理群>
       break;
}
```



switch-case文のフローチャート表現

- **goto**文も利用できるようになっている。

```
ラベル名:
   <処理群>
   goto  ラベル名;
```



goto文のフローチャート表現

【展開２】

テンプレート画面から、**if** ステートメント「If statements」と、ループ類「Loops etc...」を紹介しておく。

```
if ( "condition" ) { "statements" }

if ( "condition" ) "statement"
else "statement";

if ( "condition" ) { "body" }
else { "body" }

if ( "condition" ) "statement"
else if ( "condition" ) "statement"
else "statement"
```

```
while ( "condition" ){ "body" }

do { "body" } while ( "condition" )

repeat ( "value" ) { "body" }

for ( "init" ; "condition" ; "increment" ) { "body" }

until ( "condition" ) { "body" }

switch ( "var" ) {
  case "num" : "body"
    break;
  default : "body" }
```

このテンプレート機能を使うことで、文法的な内容のテンプレートをテキスト画面上に表示することで、コーディングのスピードアップにつながる。

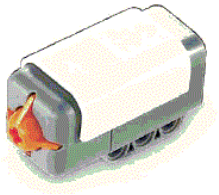
まとめ

- この章では、新たらしく「**if**」と「**do**」の２つのステートメントの**制御構造**を学んだ。
- ともに繰り返しステートメントの「**repeat**」や「**while**」を一緒に使うことで、繰り返し実行の制御を行う。
- これらは、ロボットを制御する上でも大変重要なもの。
- さらにいろいろとプログラムを編集し実行してみよう。

5. センサー

まわりの状況変化によってロボットの動きを変えるために、NXTではセンサー機能を使って実現する。

まずは、タッチ・センサーを使ったロボットを動かすために、以下の写真に示した前方にバンパーを持つ「**Tribot**」本体を組み立てる必要がある。（組立て図参照）



タッチ・センサー

【注意】このタッチ・センサーは、NXT本体下部にある**入力ポート１**に接続する。（接続ルール）

覚える事

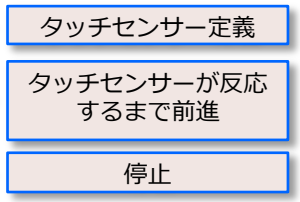
1. タッチ・センサー
2. 光センサー
3. 音センサー
4. 超音波センサー
5. untilステートメント
6. whileステートメント
7. 各種センサー設定
8. センサー値取得

センサー	標準入力ポート
タッチ・センサー	1
音センサー	2
光センサー	3
超音波センサー	4

センサーの待機（１）

それでは、ロボットが何かに接触するまで前進する簡単なプログラムを作成してみよう。

```
task main()
{
  SetSensor(IN_1, SENSOR_TOUCH);
  OnFwd(OUT_AC, 75);
  until (SENSOR_1 == 1);
  Off(OUT_AC);
}
```



【センサー設定方法1】
SetSensor(入力ポート, センサー種類)

入力ポートは、IN_1、IN_2、IN_3、IN_4のいずれか。

センサー種類は、
① タッチセンサー： SENSOR__TOUCH
② 光センサー： SENSOR__LIGHT
③ 音センサー： SENSOR__SOUND

SetSensor（接続ポート, センサー種類）；

この関数 SetSensorは、どのセンサーを利用するかをプログラム上で設定（定義）する関数。
　この行では、センサーを接続する入力ポート番号の「**IN_1**」を設定している。

他の入力ポートには、「**IN_2**」、「**IN_3**」、それに「**IN_4**」がある。
「**SENSOR_TOUCH**」は、タッチ・センサーを意味する。
光センサーの場合には「**SENSOR_LIGHT**」を使う。

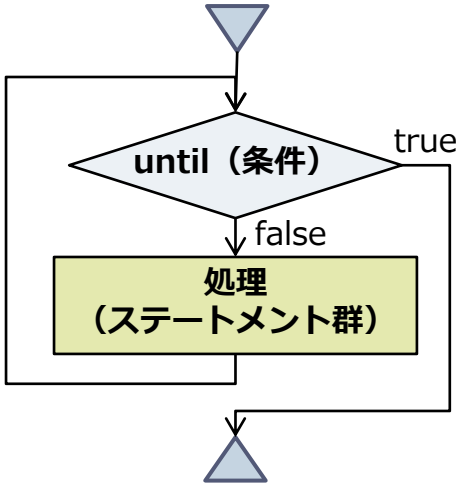
センサーの待機（２）

until(条件){ 処 理 （ステートメント群）}；

「until(条件)」の条件が真（true）の状態になるまで、次の中カッコ内を繰り返す。

つまり、ここでは設定されたセンサー「**SNSOR_1**」が「１」である状態、つまりタッチ・センサーが押されている状態になるまで繰り返す。

タッチ・センサーが「０」の状態、つまり押されていない状態では継続。すなわち、このステートメントは、タッチ・センサーが押されるまで継続（wait状態）。またスイッチを切れば、タスクは終了。



until文のフローチャート表現

タッチ・センサーの反応（１）

ロボットが障害物を避けて動くプログラムを作成してみよう。

ロボットが障害物に触れた（タッチ）ときに、ロボットを少し後退させ、方向を変え、再度前進するようにしてみよう。

```
task main()
{
  SetSensorTouch(IN_1);
  while (true)
  {
    OnFwd(OUT_AC, 75);
    if (SENSOR_1 == 1)
    {
      OnRev(OUT_AC, 75);
      Wait(300);
      OnFwd(OUT_A, 75);
      Wait(300);
    }
  }
}
```

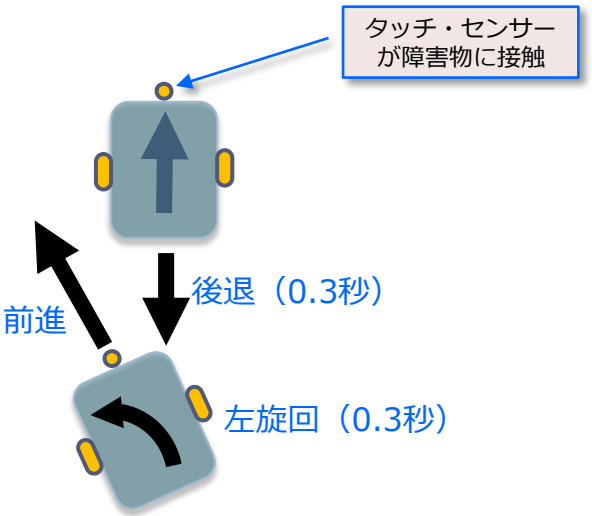
タッチセンサー設定

前進

タッチ・センサーが押されたかどうか

後退

左旋回



タッチ・センサーの反応（２）

前の例と同じように、まずセンサー定義を行っている。先ほどと違って、ここでは「**SetSensorTouch**」関数を使って、入力ポート１「IN_1」に接続している。

つぎに「while(true)」で無限ループを使って、ロボットを前進させている。

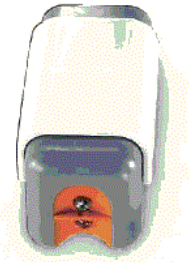
その後、タッチ・センサーが障害物に接触したかどうかを判断し、 接触した場合のみ、両方のモータで0.3秒間後退し、その後モータAを前進とモータCを後退して回転（左旋回）を0.3秒間行い、改めて両方のモータを使って前進することを行っている。

【センサー設定方法２】
① タッチセンサー：SetSensorTouch（IN_1）
② 光センサー： SetSensorLight(IN_2)
③ 音センサー： SetSensorSound(IN_3)
④超音波センサー：SetSensorLowSpeed（IN_4）

ライト・センサー【光センサー】（１）

Mindstorms NXT 2.0 の教育版購入時には、タッチ・センサーの他に、ライト・センサー（光センサー）、サウンド・センサー（音センサー）、超音波センサーが標準セットとして付属している。

＜ Mindstroms NXT 2.0 の 玩具版（商品番号８５４７）では、タッチ・センサーが２個、超音波センサーとカラー・センサー（色センサー）が各１個づつ付属している。カラー・センサーの機能は、ライト・センサーの機能を包含しているので、玩具版の場合にはこのカラー・センサーを利用する＞



光センサー

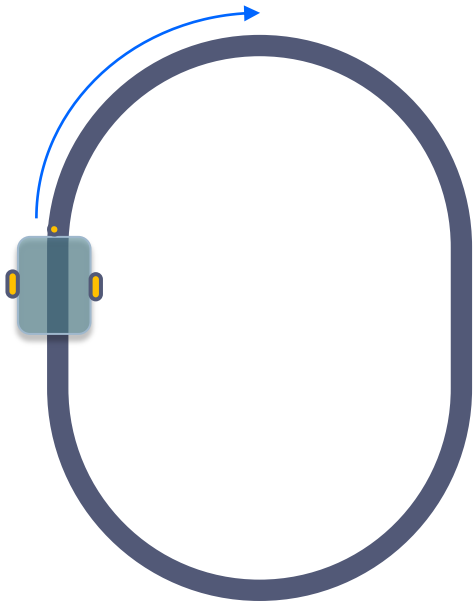
ライト・センサーにある発光LEDをオン/オフすることで、特定の方向で反射する光量や周囲の光量を調べることができる。

この反射光量を調べることで、床上のラインに沿ってロボットを動かすことができる。

つぎの事例で、どう動くかを見てみよう。

そのためにはまず「**Tribot**」ロボットに光センサーを、入力ポート３に接続する。

ここでは、NXT販売セットに付属しているテスト・パッド（黒いラインの入った大きな用紙）が必要となる。　ロボットを、黒いライン上に沿って動かすには、ライトを黒いラインの境界線上に保つことが基本となり、ライトがライン上の場合には反射光が低くなり（暗い色の場合）、黒いライン上からずれるように動かし、ライトがラインの外にある場合には反射光が高くなり（明るい色の場合）、もと（ライン上）に戻るように動かす。



付属のテスト・パッドによる
ライントレース

ライト・センサー【光センサー】（２）

```
#define THRESHOLD 40
task main()
{
  SetSensorLight(IN_3);
  while (true)
  {
    OnFwd(OUT_AC, 75);
    if (Sensor(IN_3) > THRESHOLD)
    {
      OnRev(OUT_C, 75);
      Wait(100);
      until (Sensor(IN_3) <= THRESHOLD);
    }
  }
}
```

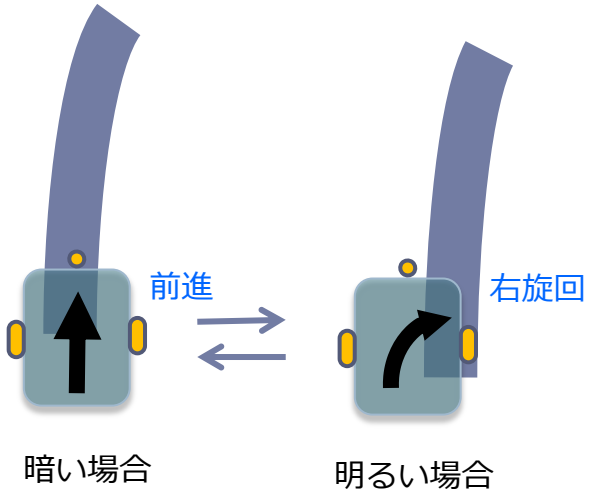
① 光センサー設定

② 前進

③明るいかどうかを判断

④ 右旋回

⑤ 暗くなるまで繰り返し



このプログラムの最初には、①「**SetSensorLight**」関数 に、入力ポート3「IN_3」を設定し、ライト・センサーを接続定義している。つぎに、② 両方のモータを前進「OnFwd（OUT_AC,75）」し、さらに「**while**(true)」 の無限ループでは、③ センサーが感知した反射光「**Sensor**(IN_3)」の値が40（定数「THRESHOLD」）以上（明るい）になると、片方のモータC の回転方向を変えてトラックに戻るまで動き続けるようにしている。④ この間0.1秒「Wait(100)」ごとに再度センサーでの反射光をチェック「**until**(・・)」し、黒いライン上に戻ったら、改めて両方のモータで前進させている。

ライト・センサー【光センサー】（３）

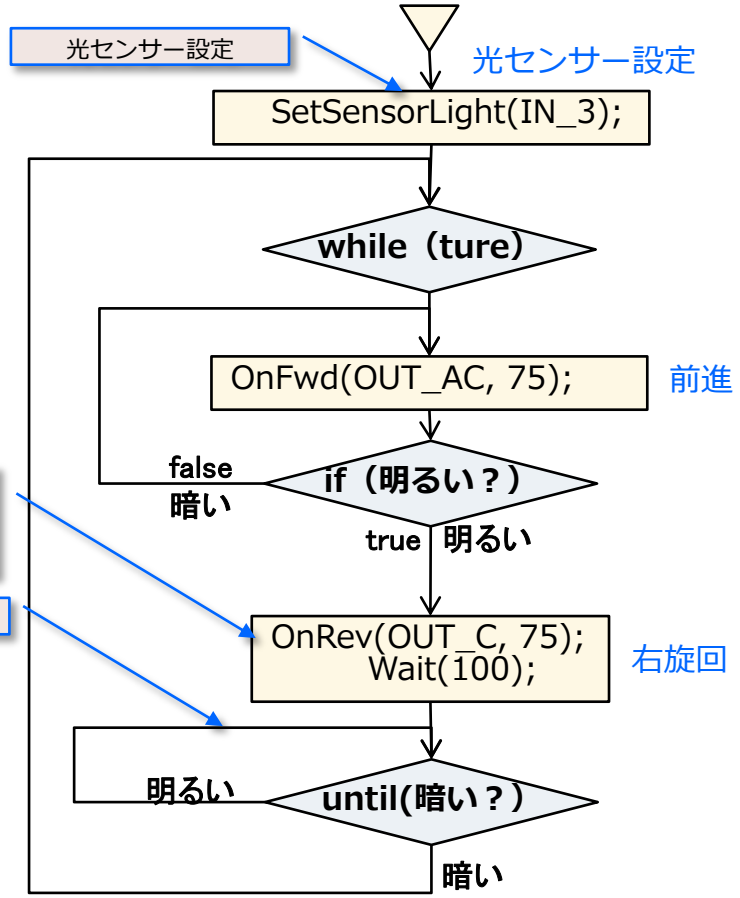
実際に、プログラムを実行すると、かならずしもライントレースとしてはスムーズには動かない。
このプログラムには、ロボットの動きで「until」 前に「Wait(100)」を入れ、モータAが前進とモータCが後退する回転を 0.1秒間 設定し、その後「until」のチェックで、黒いラインに戻るまで継続し、黒いラインの戻った時点で、つぎのステップで、再度前進「OnFwd(OUT_AC,75)」し続ける。

このプログラムでは、反時計回り（左回り）ではうまく動かない。さらに経路（黒いライン）が任意な曲線の場合には、さらに複雑なプログラムにする必要がある。

LEDライトをオフの状態でまわりの光量を計測するには、センサーをつぎのように設定する。

LEDライトを消灯

```
SetSensorType(IN_3, IN_TYPE_LIGHT_INACTIVE);
```



左旋回となるライントレース
プログラムのフローチャート

【注意】until文は、暗くなるまでループしているが、実際には、
上述のOnRev(OUT_C,75)の右旋回も同時に動いている。

サウンド・センサー【音センサー】（１）

サウンド・センサーを使い、まわりの音を感知して動くロボットをつくることもできる。大きい音を感知すれば前進し、つぎの大きい音を感知して停止し、さらに大きな音で前進・停止を繰り返すプログラムを作成してみよう。音センサーを「Tribot」の入力ポート2に接続しよう。

このプログラムの冒頭には、音量**閾値**の定数「THRESHOLD」と「SENSOR_2」の略名「MIC」を定義している。

つぎの「main」 タスクでは、まず「SetSensorSound」関数でサウンド・センサーを入力ポート２「IN_2」に設定し、「MIC」に「**SENSOR_2**」の値データが読めるように設定する。

さらに無限ループの「while(true)」に移る。

```
#define THRESHOLD 40
#define MIC SENSOR_2
task main()
{
    SetSensorSound(IN_2);
    while (true)  {
      until (MIC > THRESHOLD);
      OnFwd(OUT_AC, 75);
      Wait(300);
      until (MIC > THRESHOLD);
      Off(OUT_AC);
      Wait(300);
    }
}
```

音センサー設定

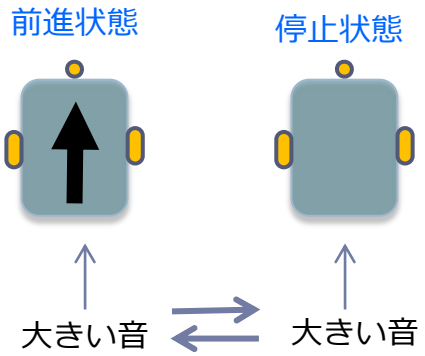
大音だと次へ

前進

大音だと次へ

停止

音センサー



【注意】この「MIC」には、サウンド・センサーで計測された音量レベルの値が変化量として入いる。

サウンド・センサー【音センサー】（２）

この無限ループ内の「until」ステートメントを使って、音量レベル（MIC）が**閾値**（THRSEHOLD）より大きくなるまで待機する。

ここでの「SENSOR_2」は、センサー名を表わすのではなく、サウンド・センサーの値を返すマクロのひとつとなる。

大きい音が発生するとロボットは動きだし、また大きい音の発生で停止し、その繰り返しを行う。

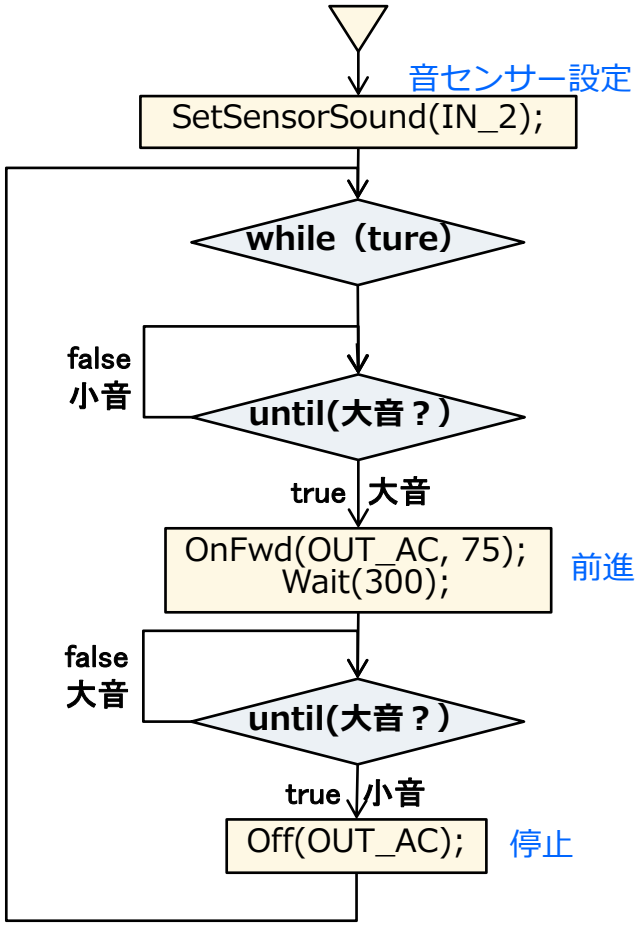
この中の「Wait」ステートメントが無い場合には、一瞬のうちにロボットは動いて止まる動作となる。実際にこの２つの「until」ステートメント間での前進と停止の実行時間もない状況となる。

また、この２つの「Wait」ステートメントをコメント行にして、どう動くかを見てみよう。

つぎに、「**until**」行の代わりに「**while**」を使ったときのステートメントは以下のようになる。

while(MIC <= THRESHOLD);

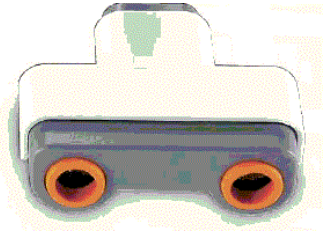
光センサーと音センサーのようなNXTのアナログ・センサーは、**SENSER_3**や**SENSOR_2**の変数値において単純な０から１００までの値のみを返す。



超音波センサー（１）

超音波センサーはソナーと同様な働きをする。概要として、超音波を発信し、障害物に当たって返ってくる超音波の反射時間を計測するものとなる。

このセンサーはデジタル・タイプで、内蔵にはデータ解析と送信のできる機能を備えている。このセンサーでは障害物を検知したり、その障害物を避けて通るようにしたりすることができる。



超音波センサー

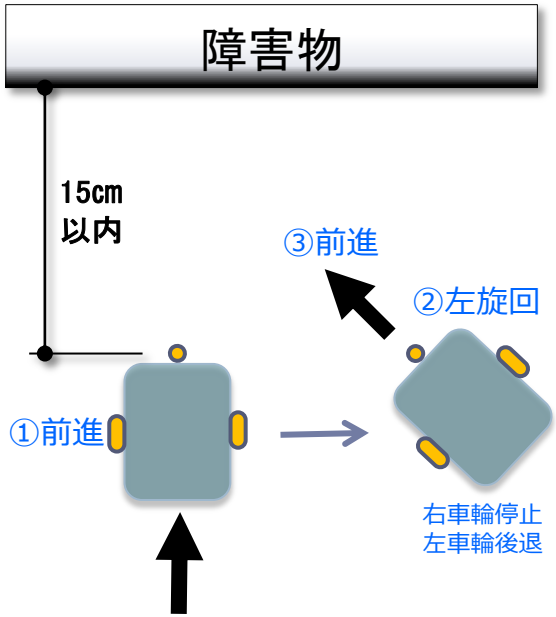
```
#define NEAR 15 //センチ単位
task main()
{
  SetSensorLowspeed(IN_4);
  while (true)  {
    OnFwd(OUT_AC, 75);
    while (SensorUS(IN_4) > NEAR);
    Off(OUT_AC);
    OnRev(OUT_C, 100);
    Wait(800);
  }
}
```

超音波センサー設定

前進

15㎝以上は継続

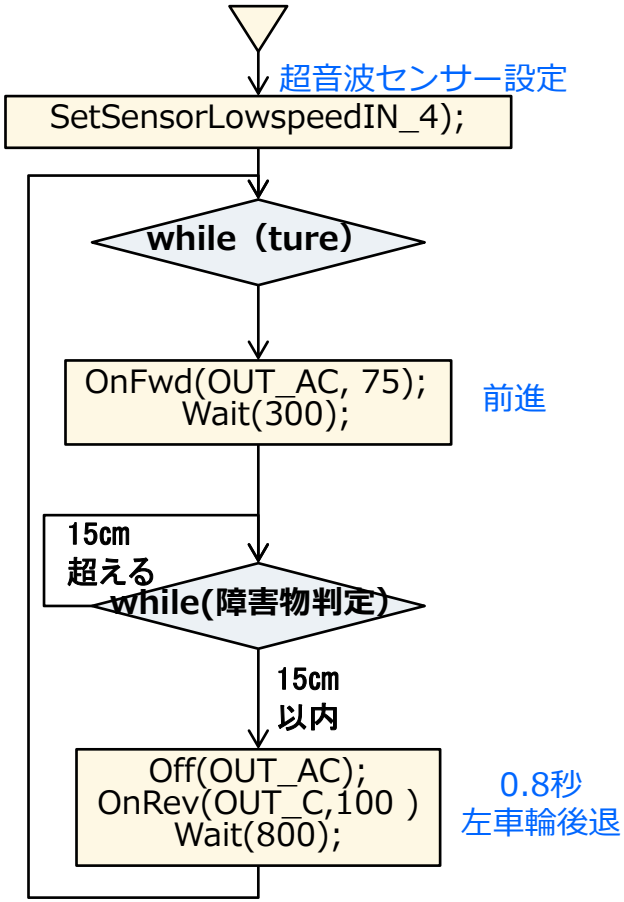
0.8秒左車輪後退



超音波センサー（２）

このプログラムでは、「**SetSensorLowspeed**」関数 で超音波センサーを入力ポート4 に設定している。

このプログラムでは、無限ループで障害物までの距離が15 センチ以下で発見されるまで前進し続ける。
15センチ以下になると、両輪を止め、0.8秒間だけモータC「OUT_C」を後進して方向を変え、
「while」ステートメントの最初の命令に戻って再び前進する。



【注意】この２つめのwhileによる障害物判定では、障害物が15cm以内になるまで、前進を継続する。つまり、前述のOnFwd(OUT_AC,75)が、中止されることなく、稼働し続けることとなる。

【展開】

ここでは、センサーに関する基本モジュールを紹介する。

モジュール	説明	利用例
SetSensor(port,const config)	センサー設定	SetSensor(S1,SENSOR_TOUCH);
SetSensorType(port,type)	センタータイプ設定	SetSensorType(S1,SENSOR_TYPE_TOUCH);
SetSensorMode(port,mode)	センサーモード設定	SetSensorMode(S 1 ,SENSOR_MODE_RAW);
SetSensorLight(port)	光センサー設定	SetSensorLight(S1);
SetSensorSound(port)	サウンドセンサー設定	SetSensorSound(S1);
SetSensorTouch(port)	タッチセンサー設定	SetSensorTouch(S1);
SetSensorLowspeed(port)	超音波センサー設定	SetSensorLowspeed(S1);
SetSensorColorBlue(port)	青色発光カラーセンサー設定	SetSensorColorBlue(S1);
SetSensorColorFull(port)	フル発光カラーセンサー設定	SetSensorColorFull(S1);
SetSensorColorGreen(port)	緑色発光カラーセンサー設定	SetSensorColorGreen(S1);
SetSensorColorNone(port)	光なしカラーセンサー設定	SetSensorColorNone(S1);
SetSensorColorRed(port)	赤色発光カラーセンサー設定	SetSensorColorRed(S1);
ResetSensor(port)	センサーの値をリセット	ResetSensor(S1);

まとめ

- この章ではNXTの標準セットに入っている全てのセンサーがどう働くかについて見てきた。
- また、「**until**」と「**while**」ステートメントの命令とセンサーの利用のしかたについても見てきた。
- もっと上達する上では、自分自身でここでのプログラムを変更・編集しロボットを動かしてみてみよう。
- より複雑な挙動をするプログラムを組込んでいくことができ、さらにNXTのロボット・センター内にあるソフトウェアをNXCプログラムに変更してみてみよう。

6．タスクとサブルーチン

ここまで紹介してきたプログラムは１つの mainタスクのみであった。

NXCプログラムでは、複数の**タスク**を持つことができる。また、プログラム内のいろいろなところで利用できるプログラムを１つの**サブルーチン**と呼ばれるプログラムで作成することができる。

これらのタスクやサブルーチンは、**プログラム全体を分かり易くする**ことができ、さらに**コンパクト（簡素化）**にすることができる。

この章では、いろいろな可能性についてご紹介する。

覚える事

1．タスク
2．サブルーチン
3．インライン関数
4．並列処理
5．排他処理
6．割り込み処理
7．マクロ処理

タスク（１）

NXC のプログラムでは、最大255 までの**タスク（task）**を持つことができ、各タスク名には重複しない（ユニーク：唯一な）名前を付ける。

また、最初に実行するタスク名には、必ず「**main**」と呼ばれる名前を付ける。

「main」以外のタスクは、他のタスクから呼ばれて実行されるか、「main」で記述されているとき実行される。

「main」タスクから実行される場合には、他のタスクがスタートする前に「main」を終了する必要がある。その時点から、両方のタスクは同時に並行して実行する。（**並列処理**）

```
task　タスク名（引数）
 {
    処理文
 }
```

一般のタスク

- タスク名はユニーク
- 最大255個まで
- 他のタスクを呼び出す
- 他のタスクで呼び出す

```
task　main（）
 {
    処理文
 }
```

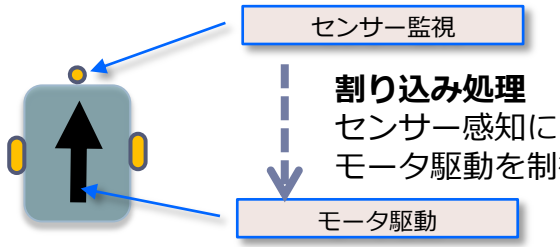
mainタスク

- mainのタスク名がメイン
- 最初に起動
- 他のタスクを呼び出す

ここで、**ロボットが四角形の軌道を描くプログラム**を作成してみよう。

ただ、**障害物があった場合には、それを避ける**ように動かす。これを一つのタスクで動かすには難しい処理となる。並列処理として、同時に２つのことをする必要がある。

１つは、モータを動かしたり止めたりして四角形軌道を動かし、もう１つはセンサーで監視すること。



並列処理
モータ駆動と
センサー監視
を同時に実行）

【注意】

並列処理：同時に2つ以上の処理を実行すること

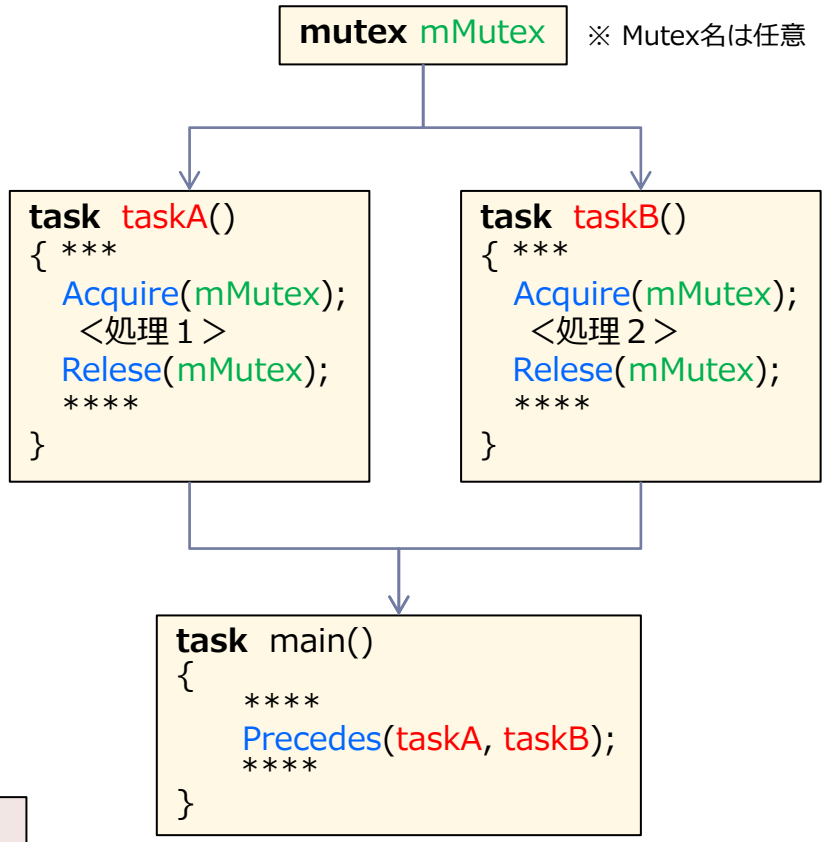
割り込み処理：障害物があったときセンサ反応で割り込み処理を実行すること

タスク（２）

このプログラム内での問題解決として、「**mutex**」（意味は「**mut**ual **ex**clusion」：相互排除）による変数「mMutex」（mutex名：任意）を定義し、「**Acquire**」と「**Release**」の２つの関数で、ある時点でのモータの動きは１つのプログラムでしか制御できないという**排他処理**を行っている。

ここでの排他処理のための変数を「**セマフォー**（semaphore:信号装置）」と呼び、この様なプログラムを**並列処理**（コンカレント）プログラムと呼ぶ。

さらに詳しくは、１０章にて述べている。



Acquire(mutex名);
＜処理＞
Relese(mutex名);

排他処理
AcquiteとReleseで囲まれた処理が優先され、他の処理は中断する。

Precedes(タスク１、タスク２、・・・タスクｎ);

並列処理
互いのタスクを並列で実行する

タスク（３）

```
mutex moveMutex;

task move_square()
{
  while (true)
  {
    Acquire(moveMutex);
    OnFwd(OUT_AC, 75); Wait(1000);
    OnRev(OUT_C, 75); Wait(500);
    Release(moveMutex);
  }
}

task check_sensors()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      Acquire(moveMutex);
      OnRev(OUT_AC, 75); Wait(500);
      OnFwd(OUT_A, 75); Wait(500);
      Release(moveMutex);
    }
  }
}

task main()
{
  Precedes(move_square, check_sensors);
  SetSensorTouch(IN_1);
}
```

並列処理を宣言

四角形を描きながら移動するタスク

並列処理開始

並列処理解放

タッチセンサの反応で動くタスク

タッチセンサ判定

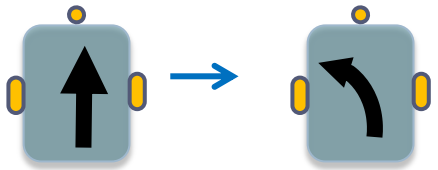
並列処理開始

並列処理解放

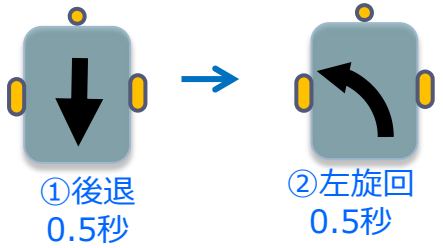
メインタスク：最初に実行するタスク

並列処理対応

タッチセンサ宣言



task move_square()



task check_sensors()

【注意】タッチ・センサーが障害物に触れたとき、check_sensorsタスクを稼働させ、move_squareタスクは中断する。

サブルーチン（１）

プログラムの中のいくつもの場所に、同じプログラム・コードを使う場合がある。このとき、**サブルーチン**（**sub**)に名前を付けたコードを組込みことができる。

このサブルーチン名を使ってタスク内で簡単に呼び出すことで、これらのコードを実行させることができる。以下に事例を使って紹介する。

```
sub　サブルーチン名（引数）
{
    処理文
}
```

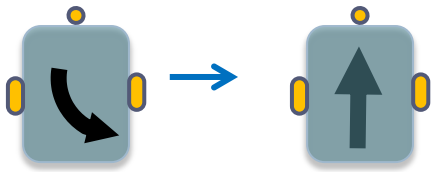
サブルーチンの定義

サブルーチン定義

引数は、ｐｗｒ：モータ馬力

```
sub turn_around(int pwr)
{
  OnRev(OUT_C, pwr); Wait(900);
  OnFwd(OUT_AC, pwr);
}

task main()
{
  OnFwd(OUT_AC, 75);
  Wait(1000);
  turn_around(75);
  Wait(2000);
  turn_around(75);
  Wait(1000);
  turn_around(75);
  Off(OUT_AC);
}
```

サブルーチンの利用



①左車輪後退
0.9秒

②前進

sub turn_around(int pwr)

このプログラムでは、ロボットをセンターまわりに回転するサブルーチン「**sub**」を定義している。

メイン・タスクでは、このサブルーチンを3回呼び出している。サブルーチンは、その名前の記述と、後に続く括弧内の数値引数をもって設定する。

もしサブルーチンに引数が無い場合には、括弧内に何も無い状態で設定する。

【注意】 サブルーチン宣言では、リターン値は無い。

サブルーチン（２）

サブルーチンの主な利点は、一つのみがNXT上で保存され、
メモリを削減できること。
しかし、サブルーチンが短い場合には、直接ライン上に
「**inline**」関数を使った方が良い場合がある。
これらは個別に保存されるのではなく、使われているとこ
ろにそれぞれ配置する。これはメモリを余分に使うが、
「**inline**」関数の制限はない。

これらは以下のように宣言する。

```
inline 型 インライン名 ( 引数群 ) {
    処理群
    return リターン値;
}
```

インライン関数

また、一般の関数は、次のように宣言する。

```
型 インライン名 ( 引数群 ) {
    処理群
    return リターン値;
}
```

一般の関数

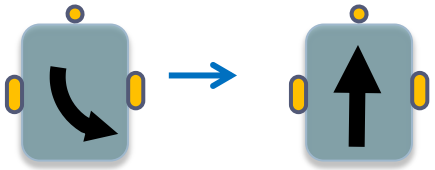
【注意】関数の型がvoidの場合は、
リターン値の宣言は不要

【注意】 関数宣言では、引数およびリターン値が
設定できる。

サブルーチン（３）

関数の引数として、回転する時間を設定することができ、以下にその例を示します

```
inline void turn_around(int pwr, int turntime)
{
  OnRev(OUT_C, pwr);
  Wait(turntime);
  OnFwd(OUT_AC, 75);
}

task main()
{
  OnFwd(OUT_AC, 75);
  Wait(1000);
  turn_around(75, 2000);
  Wait(2000);
  turn_around(75, 500);
  Wait(1000);
  turn_around(75, 3000);
  Off(OUT_AC);
}
```



①左車輪後退　②前進

inline void turn_around
(**int** pwr, **int** turntime)

インライン関数名の後の括弧内には、関数の引数群を宣言する。この例では、引数は整数型「**int**」で、その名はpwr とturntime が宣言されている。

複数の引数を宣言するときは、カンマで区切って記述する。

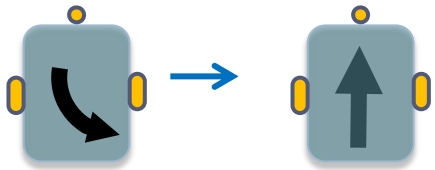
【注意】NXCでは、サブルーチン「**sub**」は「**void**」型の関数と同じで、リターン値を返さないものとなる。それに対し関数はリターン値を返すもので、整数や文字などとなる。（BricxCCのコード・エクスプローラ表示で、タスク（**Task**)とサブルーチン（**Procedure**)、関数（**Function**)の３つに区別される）

【詳細は、BricxCC内の**コード・エクスプローラ**を参照】

マクロの定義（１）

NXCでは、その他にコード名を使ってより小さくする方法として、マクロ宣言がある。すでに定数を宣言するものがマクロ宣言で、「**#define**」を使って名前を宣言する。もちろん他のコードを宣言することもできる。ここでは、前と同じプログラムとなるが、回転におけるマクロを使っている。

```
#define turn_around      ¥
  OnRev(OUT_C, 75);Wait(3400);OnFwd(OUT_AC, 75)

task main()
{
  OnFwd(OUT_AC, 75);
  Wait(1000);
  turn_around;
  Wait(2000);
  turn_around;
  Wait(1000);
  turn_around;
  Off(OUT_AC);
}
```



①左車輪後退　　②前進

#define turn_around

「**#define**」ステートメントの後に設定されている「turn_around」には、その後のテキストが定義されている。これによってその後に「turn_around」をタイプしたところには、コンパイル時にこれらのテキストが入れ替わる。

（ただ、この「**#define**」宣言されるとき、「￥」コードを使って複数行で定義できるが、分かりにくくなる難点がある。

【注意】 #define 宣言では、引数を設定できるが、リターン値は無い。

マクロの定義（２）

NXCでは、その他にコード名を使ってより小さくする方法として、マクロ宣言がある。（「BricxCC」のマクロとは異なります）すでに定数を宣言するものがマクロ宣言で、「**#define**」を使って名前を宣言します。もちろん他のコードを宣言することもできる。ここでは、前と同じプログラムとなるが、回転におけるマクロを使っている。

これらは、特に便利なマクロを宣言しています。これらは、コードをよりコンパクトにし、読みやすくしています。もちろん接続されているモータの変更などでも、より簡単に修正できます。

```
#define turn_right(s,t) OnFwd(OUT_A, s);OnRev(OUT_C, s);Wait(t);
#define turn_left(s,t) OnRev(OUT_A, s);OnFwd(OUT_C, s);Wait(t);
#define forwards(s,t) OnFwd(OUT_AC, s);Wait(t);
#define backwards(s,t) OnRev(OUT_AC, s);Wait(t);

task main()
{
  backwards(50,10000);
  forwards(50,10000);
  turn_left(75,750);
  forwards(75,1000);
  backwards(75,2000);
  forwards(75,1000);
  turn_right(75,750);
  forwards(30,2000);
  Off(OUT_AC);
}
```

【展開】

ここでは、タスクと関数との違いをを明確にしておく

	宣　言	引数	リターン値	備考
タスク(task)	task タスク名() { 処理群 }	mainのみなし。他任意	なし	並列処理、割込み処理、排他処理で利用。mainは必ず必要タスク。
サブルーチン（プロシジャ）	sub サブルーチン名（引数群） { 処理群 }	任意	なし	ほとんどタスクと同じように利用できる。[sub]の代わりに[void]を使っても同等
関数	[safecall][inline]　リターン型 関数名（引数群）{ 処理群 }	任意	リターン値任意	voidのときのみリターン値なし void のときはサブルーチンと同じ [safecall]を入れると唯一の関数のみ実行（同時並列には稼動しない） [inline]を入れると処理速度向上・ただしメモリー増大
マクロ	#define　宣言名（引数群） { 処理群 }	任意		

[注意] ここでの**引数群**の**タイプ**は、bool　および char 、 byte、 int、 short、long、 unsigned int、unsigned long、float、strings、structタイプ、およびすべてのタイプの配列となる。
また、**リターン型**には、引数群の他に、リターン値を持たないvoid宣言もある。

※ void宣言とsub宣言は、同じ意味となる。

まとめ

- この章では、タスク、サブルーチン、インライン関数およびマクロの使用について学習した。これらの使用方法はそれぞれで異なる。
- **タスク「task」**は、同時に実行するときや、異なる作業を同時に動かす時などに使う。
- **サブルーチン「sub」**は、同じタスクを異なる場所で使うコードが大きい場合に便利となる。
- **インライン「inline」関数は、**異なるタスクで、多くの異なる場所で使う場合に便利となる。ただ、メモリは増大する。
- 最後の**マクロ「define」**は、異なる場所で利用する小さいコードの場合には非常に便利。
- これらは全て引数を持つことができ、便利な使い方が理解できる。

7．ミュージックの作成

NXT は内蔵のスピーカを持ち、音を出力したり、サウンド・ファイルを再生することができる。

また特殊な使い方として、NXTに何かが発生した時、音を出して知らせるようなこともできる。

さらに、ロボットがミュージックを作成したり、動いている間に喋ったりなど、面白く工夫することもできる。

覚える事
1．音の周波数（高さ）
2．サウンド・ファイル
3．音の強さ、長さ

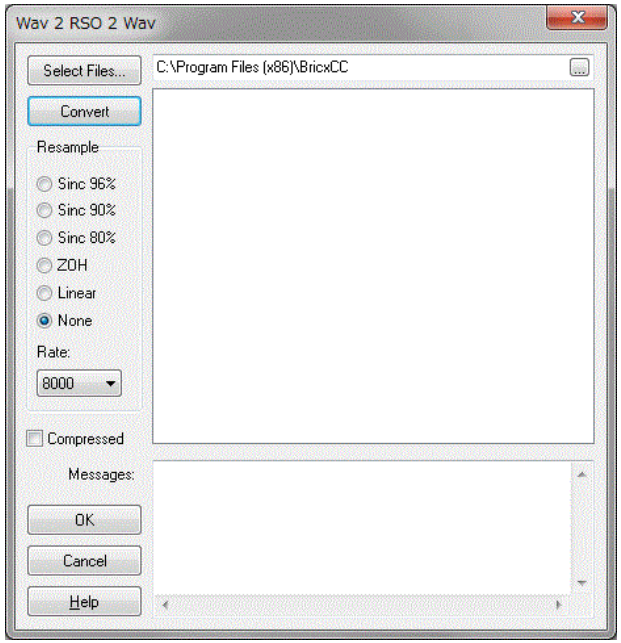
サウンド・ファイルの再生（１）

「BricxCC」は、メニューツールで、サウンド・ファイルの「.wav」ファイルを「.rso」ファイルに変換する機能を組み込んでいる（サウンド変換ツール）。

NXTのフラッシュメモリに「.rso」ファイルを読み込んで、NXTメモリに表示させ（Tools->NXTエクスプローラ）、「**PlayFileEx**」コマンドを使ってプレイすることができる。

PlayFileEx (filename, volume, loop?)

ここでの引数「**filename**」はサウンド・ファイル名、「**volume**」はボリューム（音量：０～４）、それと「**loop？**」は繰り返しの意味で、「１」（TRUE）の設定で繰り返し、「０」（FALSE）で１回のみのプレイ（再生）となる。



サウンド変換ツール
BricxCCのツールバー内の
「Tools」の「Sound Conversion」を選択

サウンド・ファイルの再生（２）

```
#define TIME 200
#define MAXVOL 4
#define MINVOL 1
#define MIDVOL  2
#define pause_4th Wait(TIME)
#define pause_8th Wait(TIME/2)
#define note_4th PlayFileEx(“! Click.rso”, MIDVOL, FALSE); pause_4th
#define note_8th PlayFileEx(“! Click.rso”, MAXVOL, FALSE); pause_8th

task main()
{
  PlayFileEx(“! Startup.rso”, MINVOL, FALSE);
  Wait(2000);
  note_4th;
  note_8th;
  note_8th;
  note_4th;
  note_4th;
  pause_4th;
  note_4th;
  note_4th;
  Wait(100);
}
```

音の強さ(0~４)

時間設定

0.2秒と0.1秒の
クリック音の定義

スタート音

リズム音

PlayFileEx(音ファイル名， 音量， 繰返し)
ここで、
　音ファイル名： rsoファイル名
　音量： 0（無音）～４（最大音）
　繰返し： true （あり）、false(なし)

このプログラムを実行させると、既にお分かりのようなリズム音が出てくる。　「タン・タ・タ・タン・タン・・・・タン・タン」
よく使うサウンドのマクロを定義し、それをmainタスクでは簡単に何度も使用している。ボリュームを変えたり、リズムを変えたりして、遊んでみよう。

ミュージックの再生（１）

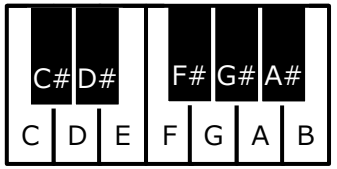
音の再生では「**PlayToneEx**(frequency, duration, volume, loop?)」コマンドを使う。

このコマンドには４つの引数があり、「**frequency**」は周波数（音の高さ）、「**duration**」は間隔（音の長さ：１/１０００秒間（ミリ秒）単位の設定で「Wait」の引数と同じ）、また「**volume**」と「**loop?**」は前述したものとなる。また「**PlayTone** (frequency, duration)」コマンドも同様に使うが、「**volume**」はNXT上のメニューで設定し、「**loop?**」は不要となる。

以下に、周波数（音の高さ）の一覧表を示す。

Sound	**3**	**4**	**5**	**6**	**7**	**8**	**9**
B	247	494	988	1976	3951	7902	
A#	233	466	932	1865	3729	7458	
A	220	440	880	1760	3520	7040	14080
G#		415	831	1661	3322	6644	13288
G		392	784	1568	3136	6272	12544
F#		370	740	1480	2960	5920	11840
F		349	698	1397	2794	5588	11176
E		330	659	1319	2637	5274	10548
D#		311	622	1245	2489	4978	9956
D		294	587	1175	2349	4699	9398
C#		277	554	1109	2217	4435	8870
C		262	523	1047	2093	4186	8372

※A（ラ）　B（シ）　C（ド）　D（レ）　E（ミ）　F（ファ）　G（ソ）



【注意】 周波数は、1秒間の振動数で、数値が大きくなるほど、高音となる。
NXCでは、周波数をそのまま音の高さとして、引数に使っている。

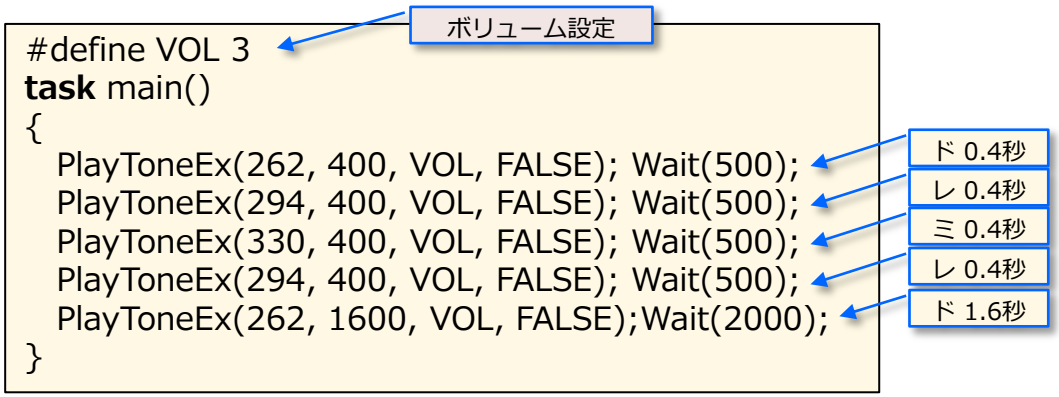
ミュージックの再生（２）

単に「**PlayFileEx**」コマンドを使った場合、サウンドは聞こえてこない。２つの音の間に「Wait」を入れて、「**PlayFlexEx**」コマンドの持続間隔を追加する必要がある。

ここでの「**PlayToneEx**」コマンドも同様に、持続間隔を追加する必要がある。

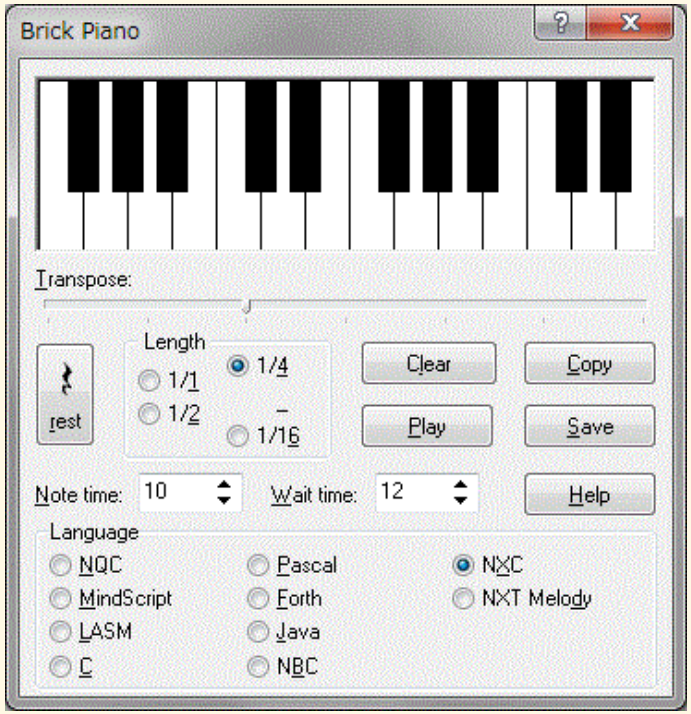
以下に使用例を紹介する。

```
#define VOL 3
task main()
{
  PlayToneEx(262, 400, VOL, FALSE); Wait(500);
  PlayToneEx(294, 400, VOL, FALSE); Wait(500);
  PlayToneEx(330, 400, VOL, FALSE); Wait(500);
  PlayToneEx(294, 400, VOL, FALSE); Wait(500);
  PlayToneEx(262, 1600, VOL, FALSE);Wait(2000);
}
```



ミュージックの再生（３）

「**BricxCC**」では簡単に**ピアン鍵盤**を用意してサウンドを作成する「Brick Piano」を用意している。NXTを動かしながら音楽を奏でるには、タスクを分けて使っていくことを推奨する。



メニューバーの「Tooles」から
「Brick Piano」を選択

つぎに、NXTを前後に動かしながら、サウンドを演奏するプログラムを紹介する。

```
task music()
{
  while (true){
    PlayTone(262,400); Wait(500);
    PlayTone(294,400); Wait(500);
    PlayTone(330,400); Wait(500);
    PlayTone(294,400); Wait(500);
  }
}

task movement()
{
  while(true){
    OnFwd(OUT_AC, 75); Wait(3000);
    OnRev(OUT_AC, 75); Wait(3000);
  }
}

task main(){
  Precedes(music, movement);
}
```

ド 0.4秒
レ 0.4秒
ミ 0.4秒
レ 0.4秒

前進3秒
後退3秒
繰返し

２つのタスク
musicと
muvementを
並列処理

【注意】Precedes関数は、並列処理を行う関数。

【展開】

ここでは、サウンドに関するモジュール群を紹介しておく。

モジュール	説明	利用例
PlayTone(frequency,duration)	音発生	PlayTone(440,500); Wait(500);
PlayToneEx(frequency,duration,volume,bLoop)	ボリューム付き音発生	PlayToneEx(440,500,2,false)
PlayFile(filename)	サウンドファイル再生	PlayFile(“Startup.rso”)
PlayFileEx(filename,volume,bLoop)	ボリューム付きサウンドファイル再生	PlayFile(“Startup.rso”,2,false)
SoundFlags()	現サウンド値を返す	X=SoundFlags();
SetSoundFlags(Flags)	現サウンドフラグを設定	SetSoundFlags(Flags);
SoundState()	サウンド状態を取得	x=SoundState();
SoundMode()	サウンドモードの取得	x =SoundMode();
SetSoundMode(mode)	サウンドモードの設定	SetSoundMode(mode);
SoundFrequency（）	サウンド周波数を取得	x=SoundFrequency（）;
SetSoundFrequency(freq)	サウンド周波数を設定	SetSoundFrequency(freq);
SoundDuration()	サウンド長さ（ミリ秒）を取得	x=SoundDuration();
SetSoundDuration(dur)	サウンド長さ（ミリ秒）を設定	SetSoundDuration(dur);
SoundVolume()	サウンドボリュームの取得	x=SoundVolume();
SetSoundVolume(volume)	サウンドボリュームの取得	SetSoundVolume(volume）;

まとめ

- この章では音やサウンドを作成することを学習した。
- またサウンドを別タスクとして利用することも見てきた。
- いろいろと音とサウンドを使うことで、制御の分岐点に入れることで、ロボットの動きを分かり易くすることができる。
- また、センサーの反応に応じた量を、音やサウンドで知らせることもでき、プログラムの動きを知ることもできる。

8．モータの詳細設定

モータをより精密に制御するためのコマンドは、他にもいろいろと持ち合わせている。

この章では「**ResetTachoCount**」、「**Coast**」、「**Float**」、「**OnFwdReg**」、「**OnRevReg**」、「**OnFwdSync**」、「**OnRevSync**」、「**RotateMotor**」、「**RotateMotorEx**」、および基本的な「**PID制御**」の 概念について学習していこう。

覚える事

1．モータの詳細設定
2．減速停止
3．PID制御
4．同期モード

減速停止

「**Off()**」コマンドを使った場合、サーボモータは即座に停止し、車軸も止まり、位置を固定する。

NXTのサーボモータは、減速してゆっくりと停止することもできる。この場合、「**Float()**」や「**Coast()**」コマンドはそれぞれ使って、ゆっくりとモータの力を簡単に停止できる。

ここでプログラム例を紹介しよう。最初は、「**Off()**」コマンドを使って即座に停止させ、つぎに「**Float()**」コマンドを使って、減速させて停止させている。

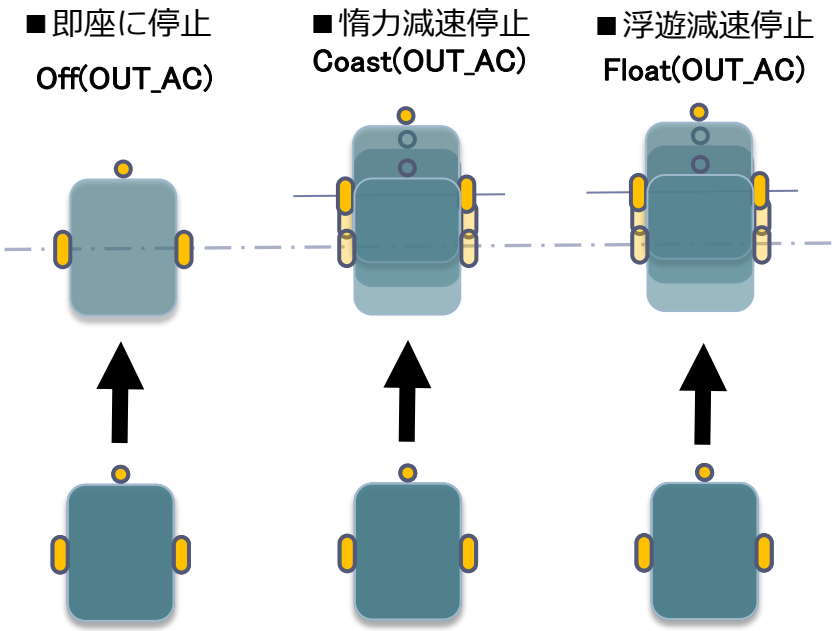
この違いを理解してみよう。ここでのロボットの停止の違いは微妙であろう。他のロボットを用いてテストした場合には、違いが明確になることがあります。

この「**OnFwd()**」と「**OnRev()**」コマンドは、モータを動かすための基本的なルーチンとなる。

```
task main()    //  Program8.1
{
  OnFwd(OUT_AC, 75);
  Wait(500);
  Coast(OUT_AC);
  Wait(3000);
  OnFwd(OUT_AC, 75);
  Wait(500);
  Float(OUT_AC);
}
```

減速停止

減速停止



【注意】Coastの惰力減速停止とFloatの浮遊減速停止の違いはほとんどない。

上級コマンド（１）

NXTのサーボモータは、車軸の回転やスピードを正確に制御するための機能を持ち合わせていて、NXT内蔵の**ファームウェア**には、モータ駆動でのフィードバックを行う**PID制御**（Proportional Integrative Derivative：後述）を実装している。

もし、ロボットを正確に前進させるとき、両方のモータの同期をとるよう、つまり一方のモータが遅れたり、互いに遅くなったりしてもモータの動きを調整して同じになるようにこの制御を使う。

さらに、右回転や左回転では、互いのモータのパワーにパーセントを使って同期をとって制御したりすることもできる。 これらのようにサーボモータをフルに活用するための多くのコマンドがある。

「**OnFwdReg**(ports, speed, regmode)」コマンドでは、「**ports**」で出力ポートを、「**speed**」で速度を、それに「**regmode**」で調整モードを指定し、モータを制御します。ここでの「**regmode**」調整モードでは、**OUT_REGMODE_IDLE**（「IDLE」モード）、**OUT_REGMODE_SPEED**（「SPEED」モード）、および**OUT_REGMODE_SYNC**（「SYNC」モード）を使用する。この「**IDLE**」モードは、PID 制御なしでの状態で、つぎの「**SPEED**」モードは、１つのモータがどんな場合でも一定速度で動くように調整する状態、それに「**SYNC**」モードは、前に述べたように同期をとって２つのモータを動かす状態（**同期モード**）のことを意味する。

「**OnRevReg()**」は、前述したコマンドの逆方向へ動かすものとなる。

OnFwdReg(ports, speed, regmode);　　**OnRevReg**(ports, speed, regmode);
ここで、各パラメータは以下のとおり
ports: **出力ポート**
speed: モータの**パワー**（０～100）
regmode: **調整モード**
　IDLEモード： PID制御なし
　SPEEDモード ： １つのモータがどんな場合でも一定速度で動くように調整
　SYNCモード　： 同期をとって２つのモータを動かす（**同期モード**）

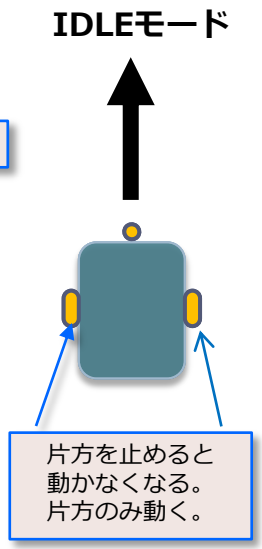
上級コマンド（２）

```
task main()    //  Program8.2
{
  OnFwdReg(OUT_AC, 50, OUT_REGMODE_IDLE);
  Wait(2000);
  Off(OUT_AC);
  PlayTone(4000, 50);
  Wait(1000);
  ResetTachoCount(OUT_AC);
  OnFwdReg(OUT_AC, 50, OUT_REGMODE_SPEED);
  Wait(2000);
  Off(OUT_AC);
  PlayTone(4000, 50);
  Wait(1000);
  OnFwdReg(OUT_AC, 50, OUT_REGMODE_SYNC);
  Wait(2000);
  Off(OUT_AC);
}
```

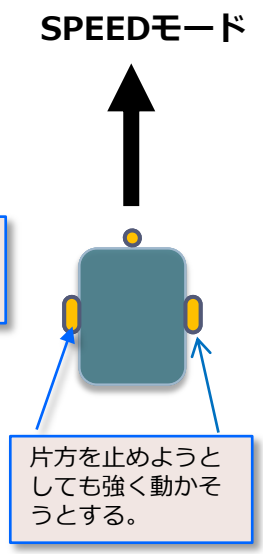
IDELモード

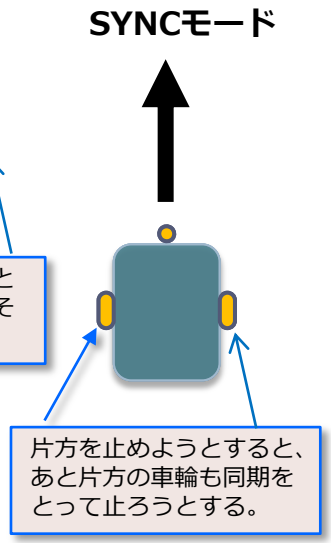
SPEEDモード

SYNCモード



プログラム実行中に、片方の車輪を手で止めてみよう。もう一方の車輪がどう反応するか？





このプログラムを、ロボットの両方のモータが回転しないように手で止めた状態で動かしてみよう。
最初の「**IDLE**」モードでは、手の力の入れ方によって止めた**片方のモータが動かなくなる**状態になり、手で止めてないモータは動いた状態となる。
つぎの「**SPEED**」モードでは、手で止めようとしても一定にモータを**強く動かす**ような状態になる。
最後の「**SYNC**」モードでは、１つのモータを手で止めようとすると、片方のモータも止まるように、両モータが**同期をとる状態**になる。

上級コマンド（３）

「**OnFwdSync**(ports, speed, turnpct)」コマンドは、「**OnFwdReg**()」コマンドの「**SYNC**」モードと同様に、両輪が同期を取って動く。

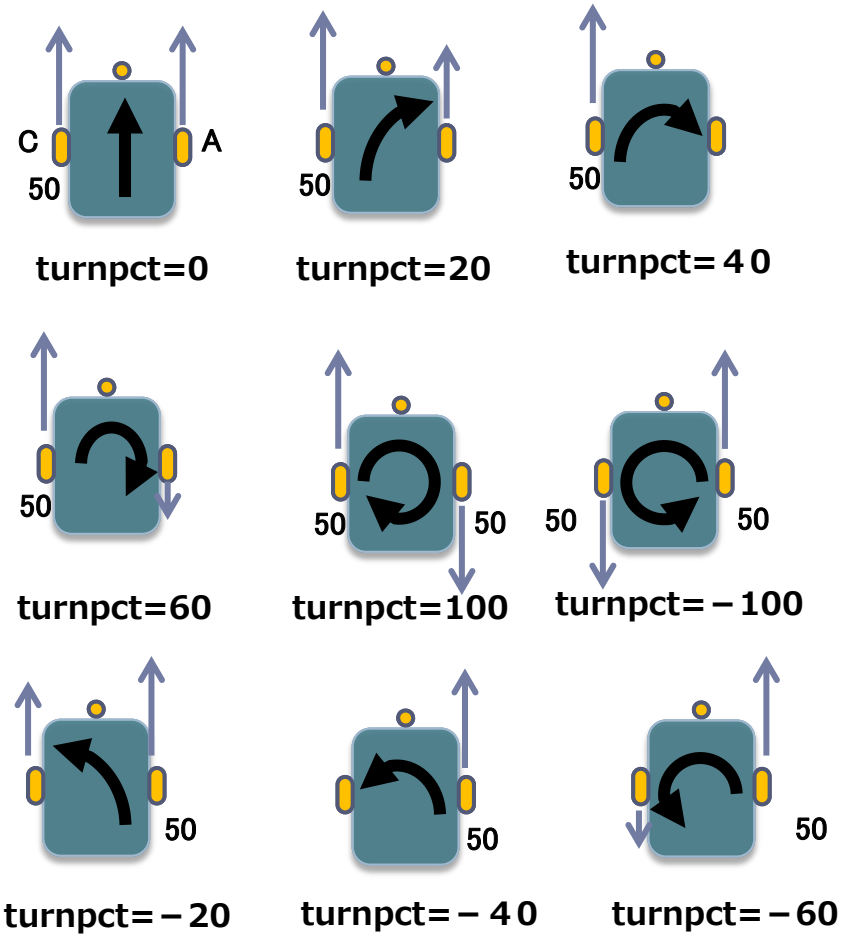
しかし、新たな引数「**turnpct**」の設定によってスピードをパーセンテージ（**パワー比**：－１００から１００まで）で調整できる。

「**OnRevSync**()」は、同様にモータを逆方法へ動かすもとなる。

つぎのプログラムは、これらのコマンドを使っています。いろいろとパラメータを変更し実行させてみてください。

```
task main()     //  Program8.3
{
  PlayTone(5000, 30);
  OnFwdSync(OUT_AC, 50, 0);
  Wait(1000);
  PlayTone(5000, 30);
  OnFwdSync(OUT_AC, 50, 20);
  Wait(1000);
  PlayTone(5000, 30);
  OnFwdSync(OUT_AC, 50, -40);
  Wait(1000);
  PlayTone(5000, 30);
  OnRevSync(OUT_AC, 50, 100);
  Wait(1000);
  Off(OUT_AC);
}
```

Turnpctパラメータを変更してためしてみよう

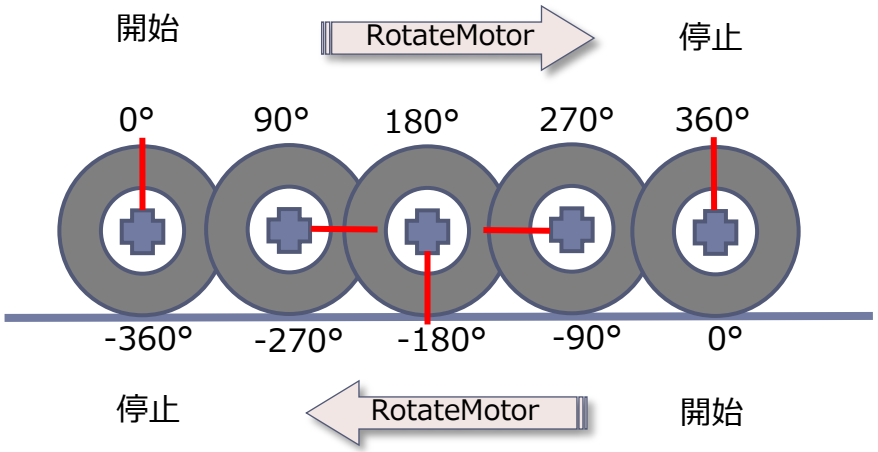


【注意】実際の動きは、床の材質の状態によって大きく異なる。

上級コマンド（４）

つぎに、モータを**回転角度**で制御する関数を紹介しよう。
つぎの両方のコマンドは、符号をつけた回転角度使ってモータを動かしている。マイナスを付けたモータは、逆方向へ回転することになる。　（１回転は３６０度）

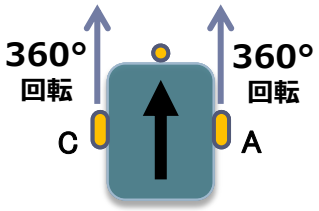
「**RotateMotor**(ports, speed, degrees)」の引数の「**ports**」は出力ポート、「**speed**」はスピード（**パワー**：０～１００）、それと「**degrees**」は**回転角度**となる。



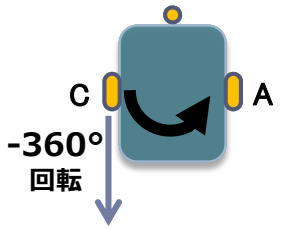
駆動するモータの回転角度を正確に制御

```
task main()      // Program8.4
{
  RotateMotor(OUT_AC, 50, 360);
  RotateMotor(OUT_C, 50, -360);
}
```

【注意】RotateMotorの後にWaitによる待機を使っていないことに注意。つまり、指定された回転角度を実施するまで稼働し続けることでWaitは不要となる。



RotateMotor(OUT_AC, 50, 360);



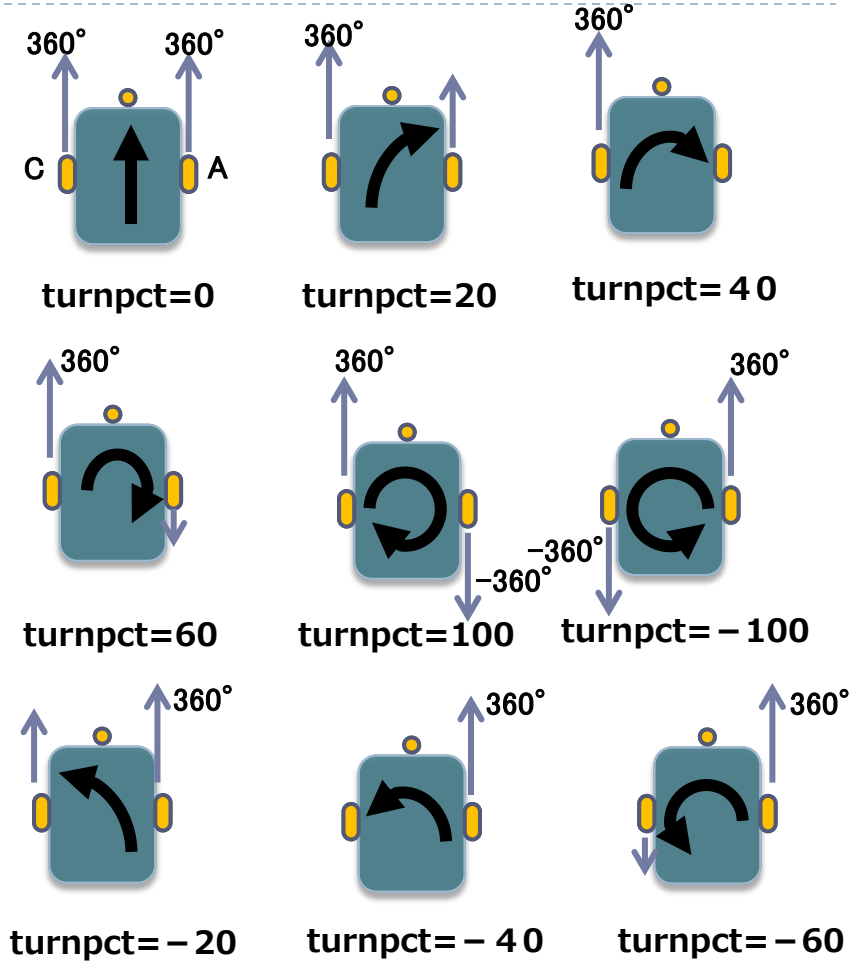
RotateMotor(OUT_C, 50, -360);

上級コマンド（５）

「**RotateMotorEx**(ports, speed, degrees, turnpct, sync, stop)」は、上述の拡張コマンドで、両方のモータ（例えばOUT_AC）の同期をとって動かすもので、新たな引数「**turnct**」は**パワー比**（－１００～１００）で、「**sync**」と「**stop**」は共に真偽値（真＝１、偽＝０：ブーリアン変数）となる。
このパワー比は、０の場合には両輪とも同じパワーで前進し、100と―100の場合には、両輪が真逆に動くことになる。
「sync」は、両輪の同期取るモード（真）で、同期を取らないモード（偽）で、「**stop**」は回転角度を正確に止めるかどうかの引数となる（真＝1：即停止 、偽＝0：減速停止）。

```
task main()    //  Program8.5
{
  RotateMotorEx(OUT_AC, 50, 360, 0, true, true);
  RotateMotorEx(OUT_AC, 50, 360, 40, true, true);
  RotateMotorEx(OUT_AC, 50, 360, -40, true, true);
  RotateMotorEx(OUT_AC, 50, 360, 100, true, true);
}
```

【注意】RotateMotorExの後にWaitによる待機を使っていないことに注意。つまり、指定された回転角度を実施するまで稼働し続けることでWaitは不要となる。



【注意】実際の動きは、床の材質の状態によって大きく異なる。

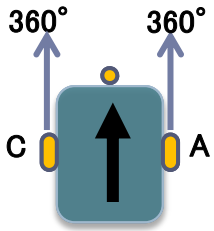
PID 制御（１）

NXT のファームウェアでは、デジタルによる**PID制御**（proportional integrative derivative：比例・積分・微分）で、**サーボモータの位置とスピードを正確に制御**することができます。この制御方式は、より効果的な閉じたループの**フィードバック制御**の中の最も簡単な一つとなる。

簡単に言えば、ある値から、定値の定温度や定速度になるまで温度（速度）を上下させるとき、現値と定値の誤差による計算（P）、時間変化での値を、積分による計算（I）や微分の計算（D）によって、コントロール（制御）する方法となる。

プログラムでは、現在位置から達成値（定値） R(t) までに移動させるために、 モータ動作を指令 U(t) し、組み込んだ**エンコーダ**で新たな現在位置（現値） Y(t) を計測して、その達成値との差（誤差）を E(t)＝R(t)-Y(t)　で求める。

ここで「**閉ループ制御**」は、計測された現在位置 Y(t) を新たな誤差計算に使っていることなる。この制御では、ここで求まった誤差 E(t)を、モータを動かすための指令U(t)に変換する。



PID 制御（２）

そこで

```
U(t) = P(t) + I(t) + D(t), ここで
P(t) = Kp・E(t)
I(t) = Ki・( I(t+1) + E(t) )  および
D(t) = Kd・( E(t) - E(t-1) )
```

この式は、初心者にとっては難しく見えるが、簡単に紹介しておく。

この **U(t)** は、３つの式の総和でなりたち、それぞれ**比例（偏差）部分**が「**P(t)**」、**積分部分**が「**I(t)**」、それに**微分部分**が「**D(t)**」となる。

「**P(t)**」は、装置(モータ)を瞬時に動かすもので、誤差がゼロにすることはできない。

「**I(t)**」は、制御において記憶装置を持ち、誤差変動を計測して、補正し、ゼロまでの安定状態を行う。

「**D(t)**」は、制御において予測装置を持ち、高速な応答を（微分計算で）実現している。

PID制御とは、フィードバック制御の一種。入力値の制御を、その偏差（誤差）の比例、積分、それに微分の３つの要素によって行う方法。きめ細かな制御ができ、自動制御方式の中でもっともよく使われる制御方式。

PID 制御（３）

ここではNXTを接続して、具体的にどう動くか見てみよう。
つぎの簡単なプログラムを使って学習してみよう。

```
#define P 50    //  Program8.6
#define I 50
#define D 50

task main()  {
  RotateMotorPID(OUT_A, 100, 1800, P, I, D);
  Wait(3000);
}
```

この「**RotateMotorPID**(port, speed, angle, Pgain, Igain, Dgain)」を使い、この中の PID値を変更させ、モータを動かしてみよう。

例えばつぎのように値を設定してみてみよう。

(50, 0, 0) ：モータは補正なしの誤差が残るため正確に1800度（タイヤが5回転）の回転をしない。

(0, x, x) ：比例制御が無いため誤差は大きくなる。

(40, 40, 0) ：目標位置を超え戻ろうとする。

(40, 40, 90) ：精度よく起動する（時間は設定立ち上がり時間（指定位置に着くまでの時間）

(40, 40, 200) : 微分のゲインは多すぎるため車輪は不安定に動く。

他の値も変更してモータの反応にどう影響するかを試してみよう。

＜**補足説明**：本プログラムのコンパイル時にファームウェアエラーが表示された場合には、BricxCC画面上のメニューバーの「Edit」から、環境設定「Preferecnes…」を選択し、「Preferences」画面上の「Compiler」タグのなかのさらに「NBC/NXC」タグを選択します。この中の「Enhanced firmware」（拡張ファームウェア）をクリック（選択）し、「OK」ボタンで終了させ、BricxCCを再起動して再コンパイルしてみよう＞

【展開】

ここで使ったAPI　出力関数モジュール群をまとめておく。

モジュール	引　数	説明
ResetTachoCount	出力ポート	タコメータ（回転速度計）カウンタのリセット
Coast	出力ポート	モータ停止、惰力走行
Float	出力ポート	モータ停止、浮遊走行（Coastと同じ動き）
OnFwdReg	出力ポート、パワー、調整モード	調整モードに合わせて前進
OnRevReg	出力ポート、パワー、調整モード	調整モードに合わせて後退
OnFwdSync	主力ポート、パワー	両輪同期を取った調整モードに合わせた前進
OnRevSync	主力ポート、パワー	両輪同期を取った調整モードに合わせた後退
RotateMotor	出力ポート、パワー比、回転角度	回転角度の調整によるモータ制御
RotateMotorEx	出力ポート、パワー比、回転角度，同期モード、停止モード	回転角度の調整によるモータ制御（拡張機能）
RotateMotorPID	出力ポート、パワー、回転角度、P値、I値、D値	PID制御（詳細は後述）

【補足】 モータの**パワー**は0～100の値、
調整モードはOUT_REGMODE_IDLE=0,OUT_REGMODE_SPEED=1,OUT_REGMODE_SYNC=2,OUT_REGMODE_POS=4のいずれか、
回転角度は 整数値（負の値も含む）、 **同期モード**と**停止モード**は共にブーリアン値（1:真、0:偽）、 **P値、I値、D値**は0～255の値

まとめ

- この章では、高度なモータ制御用のコマンド群を学んだ。
- 「**Float()**」と「**Coast()**」コマンドは、モータを減速停止する機能
- 「**OnXxxReg()**」と「**OnXxxSync()**」コマンドは、モータのスピードと同期をフィードバック制御する機能
- さらに「**RotateMotor()**」と「**RotateMotorEx()**」コマンドは、正確に車軸を回転角度で制御する機能
- さらにPID制御についても学んだ。（詳細は、Webなどで）

9．センサーの応用について

５章では、センサー利用の基本的な特徴について述べた。ここではさらにセンサーのさまざまな可能性について詳しく紹介していく。

この章では、センサーのモードとタイプについての違いについて学習し、古いコンパチブルのRCXセンサーをLEGO変換ケーブルを使ってNXTに接続することを学ぶ。

覚える事

1．センサーモードとタイプ
2．モード設定
3．タイプ設定
4．I2Cセンサー

センサーのモードとタイプ（１）

前述の「**SetSensor**()」コマンドでは、入力ポートとセンサータイプの２つのパラメータで設定する方法を学んだ。

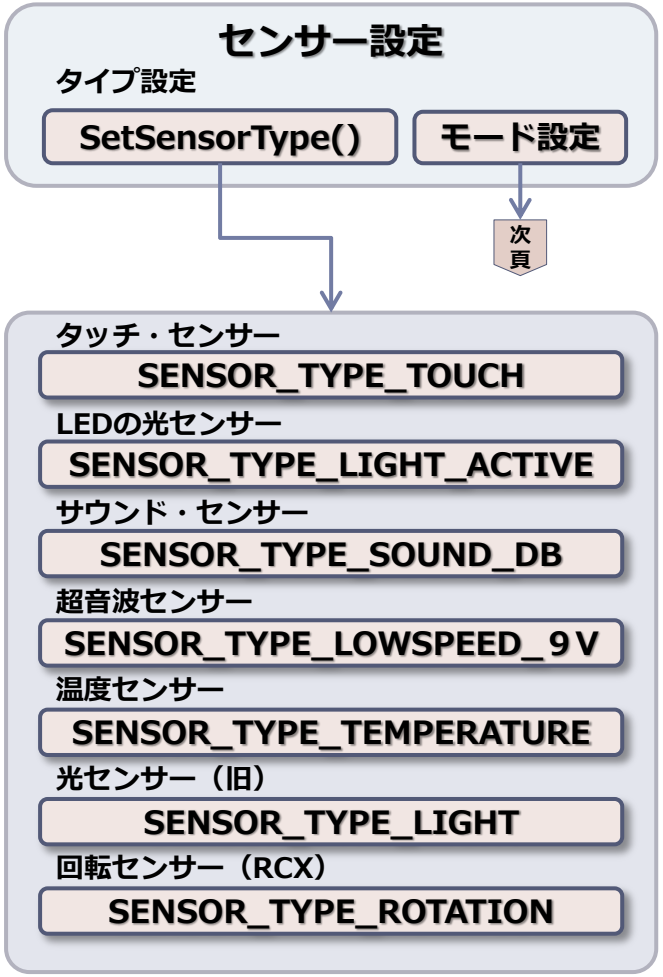
ここでは、センサーの**タイプ設定**と、センサーの働きの**モード設定**を行う方法を学ぶ。

このセンサー設定をそれぞれモードとタイプに分けて設定することで、特殊なアプリケーションでのセンサーの制御を、より使いやすく、より正確に行うことができる。

タイプ設定は「**SetSensorType**()」コマンドで、多くのタイプの設定を行うことができる。

ここでは主なものを紹介する。タッチ・センサーは「**SENSOR_TYPE_TOUCH**」で、LEDの光センサーは「**SENSOR_TYPE_LIGHT_ACTIVE**」、サウンド・センサーは「**SENSOR_TYPE_SOUND_DB**」、それと超音波センサーは「**SENSOR_TYPE_LOWSPEED_9V**」となる。

また**タイプ設定**では、センサーの**パワー**（例えば、LEDの光センサーの光量など）の指示や、デジタル通信で**I2C**センサープロトコルを使うような、NXTへの指示が必要なとなどがある。また、古いRCXのセンサーをNXTで使うように設定することもでき、温度センサーの「**SENSOR_TYPE_TEMPERATURE**」、古い光センサーの「**SENSOR_TYPE_LIGHT**」、RCXの回転センサーの「**SENSOR_TYPE_ROTATION**」（別途後述）などがある。



【補足説明】I2C は**I**nter-**I**ntegrated **C**ircuit （I-squeqrd C）の略で、低速な周辺機器用の接続シリアルバス（フィリップス社開発）

センサーのモードとタイプ（２）

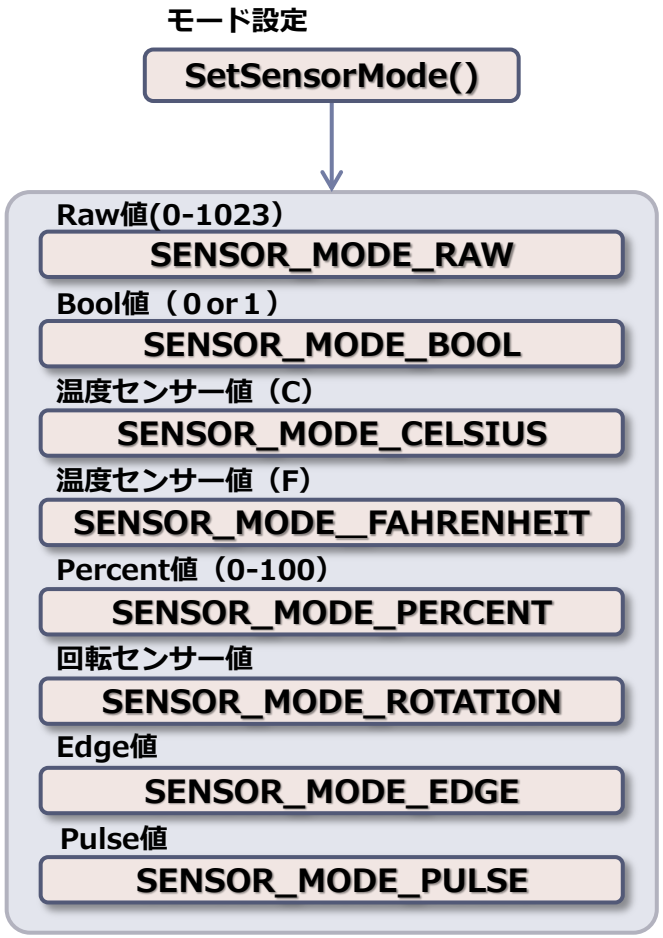
センサーの**モード設定**は「**SetSensorMode**()」コマンドでもできる。モード設定には8種類ある。

最も重要なひとつに「**SENSOR_MODE_RAW**」がある。このモードでは、センサーの値を 0 から 1023 の間の数値で確認できる。つまり、センサーの未処理の値（raw value）を出力する。

現在のセンサーの値に依存したものとなる。例えば、**タッチ・センサー**では、何も押されていない状態での値は1023に近づく。また確実に押された状態での値は50に近づく。部分的に押された状態での値は 50～1000の値となる。この様に、タッチ・センサーのモード設定で未処理の値を使うことで、部分的に押されている状況かどうかを知ることができる。

光センサーの場合には、300（非常に明るい）から 800（非常に暗い）の値を示す。

このコマンドは「**SetSensor**()」コマンドよりも、もっと正確な値を返すことになる。詳細については、NXCプログラミングガイドを参照してほしい。



センサーのモードとタイプ（3）

つぎに紹介するセンサー・モードは「**SENSOR_MODE_BOOL**」です。このモードでは、 0 か 1 かの値となる。

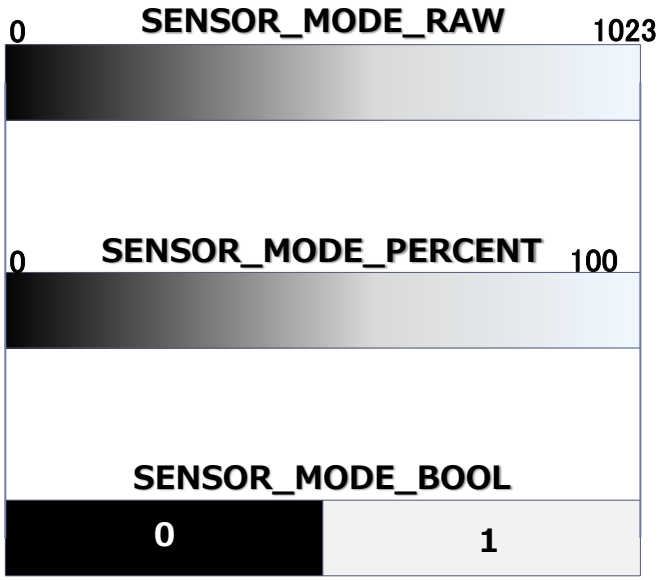
上で説明した未処理の値（raw value）が 562 よりも上の場合には 0 になり、その他は 1 となる。

この「**SENSOR_MODE_BOOL**」はタッチ・センサーの省略値だが、他のセンサーでもアナログデータを簡略化したものとして利用できる。

また「**SENSOR_MODE_CELSIUS**」（温度値「°C」）と「**SENSOR_MODE_FAHRENHEIT**」（温度値「F」）のモードは温度センサーのみで利用でき、互いのモードの温度値を返す。

さらに「**SENSOR_MODE_PERCENT**」モードは未処理の値を 0 から 100 までの値を返す。このモードは光センサーの省略値のモードとなる。

それから「**SENSOR_MODE_ROTATION**」モードは、回転センサーのみに利用される。（後述）



センサーのモードとタイプ（４）

他に２つの興味あるモードに「**SENSOR_MODE_EDGE**」と「**SENSOR_MODE_PULSE**」がある。

これらは変化量を測り、未処理の値が小さい値から大きい値への変化やその逆の値の変化値となる。

例えば、タッチ・センサーに触れたとき、大きい値から小さい値への未処理の値に変化する。また手を離した場合には、逆な方向に値が変わる。

「**SENSOR_MODE_PULSE**」モードを設定した場合には、小さいほうから大きい値のみの変化量しか出力しない。それでタッチ・センサーに触れて離した場合は１回のカウントとなる。

「**SENSOR_MODE_EDGE**」モードを設定した場合には、両方への変化でのカウントを行う。よってタッチ・センサーに触れて離した場合には２回カウントされる。これによって、何回タッチ・センサーが押されたかを知ることができる。もしくは光センサーとの組み合わせで、何回ライトのスイッチをオン・オフしたかのカウントにも利用できる。

もちろんこのモードを使う場合には、カウンタを 0 に戻すことも必要で、「**ClearSensor()**」コマンドを使って、計測したセンサーのカウンタを消去することができる。

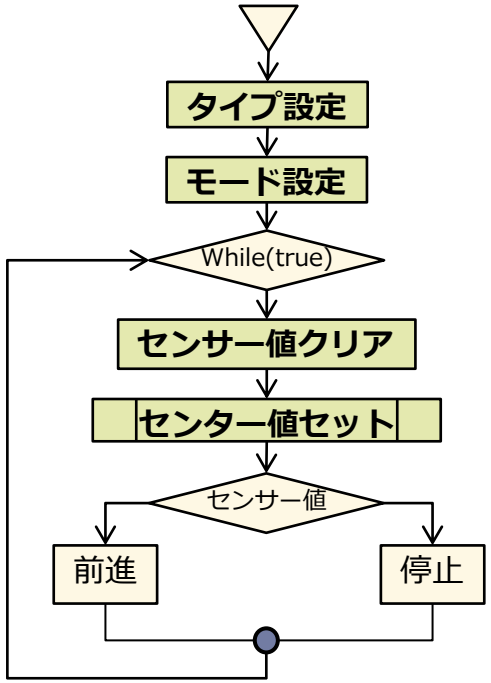
センサーのモードとタイプ（５）

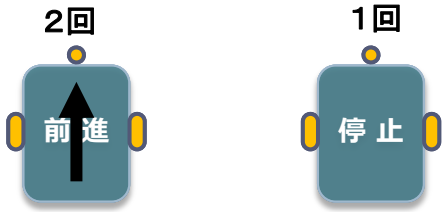
それでは事例を使って学習していこう。つぎのプログラムはタッチ・センサーを使ってロボットを動かしている。タッチ・センサーを長いケーブルでNXTの入力ポートのひとつに接続してみよう。瞬時に２度タッチ・センサーに触れた場合には、ロボットは前進する。もし１度だけだと動きを止めるようになっている。

```
task main()
{
  SetSensorType(IN_1, SENSOR_TYPE_TOUCH);
  SetSensorMode(IN_1, SENSOR_MODE_PULSE);
  while(true)
  {
    ClearSensor(IN_1);
    until (SENSOR_1 > 0);
    Wait(500);
    if (SENSOR_1 == 1) {Off(OUT_AC);}
    if (SENSOR_1 == 2) {OnFwd(OUT_AC, 75);}
  }
}
```

タイプ設定
モード設定
センサー値クリア
停止
前進

【注意】ここで重要な点として、最初にセンサーのタイプを設定し、つぎにモードの設定を行うことである。このことは基本的なことで、センサー・タイプの変更は、モードに影響することになる。





Wait(500）で、0.5秒間にタッチセンサーが２度押されると「前進」、１度押されると「停止」する

回転センサー（１）

回転センサーは非常に便利な機能で、NXTのサーボモータに内蔵された**光学エンコード**で、相対的な角度位置が計測でき、車軸穴を持っている。正確な**１回転を１６段階**（もしくは逆方向だと－１６）でカウントし、**１段階を 22.5 度**の回転で、サーボモータが持つ1度ごとの回転に比べれば荒くなっている。

以前からある回転センサーは、モータを回転させることなく、車軸の回転を表示させ、モータを動かすトルク（回転モーメント）からモータを利用することがでる。以前の回転センサーでは簡単にモータを回転できる。

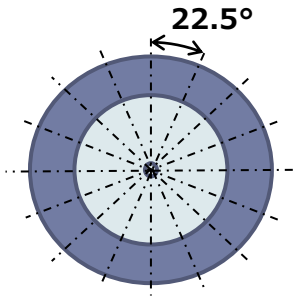
もし、この１６段階でモータの回転を利用する場合には、１回転ごとこの数値を増やすことで、いつでも機械的にギアを利用することができる。

標準的なアプリケーションでは、２つモータ制御をもつロボットでは、２つの軸に接続した２つの回転センサーを持つことになる。

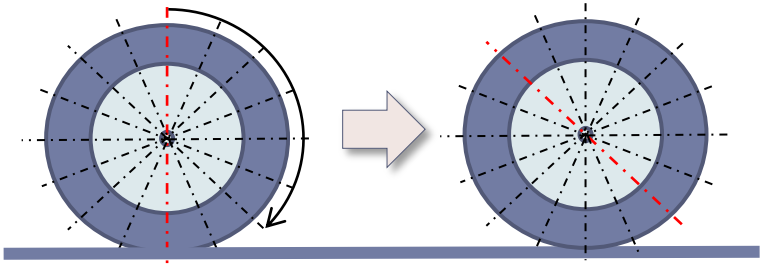
まっすぐにロボットを動かすには、両方の車輪を同等に回転させる。しかしながら、モータは必ずしも正確に同じスピードでは動かない。

この回転センサーを使うことで、１つの車軸を高速に回転できる。

両方の回転センサーが、同じ値になるまで、一時的にモータを停止（「**Float**()」を使って）させることができる。



回転センサーを利用することで、1回転を16分割（1分割=22.5度）で、制御可能



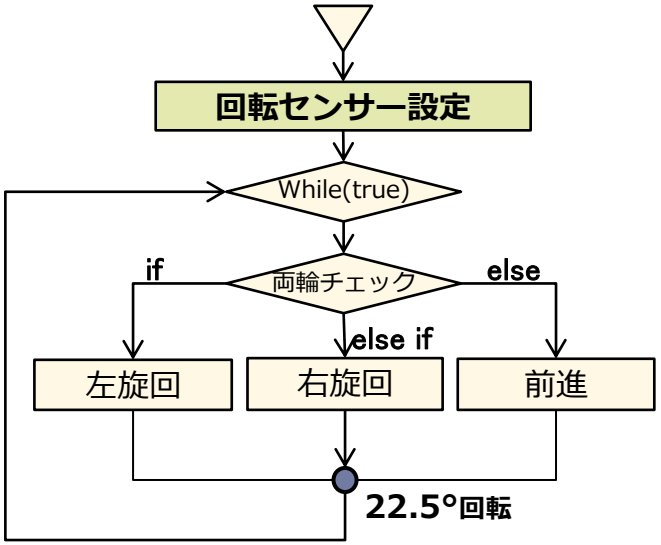
両輪でのモータ駆動を正確に行うことができ、直線での前進が可能となる。

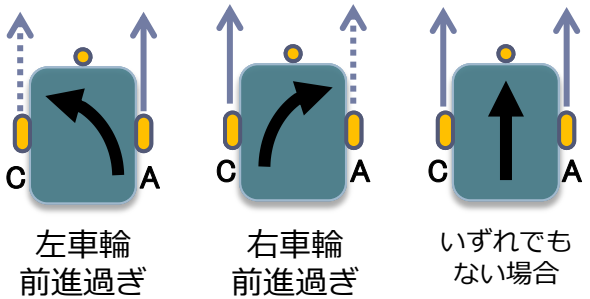
回転センサー（２）

つぎのプログラムでは、回転センサーを使った事例である。

このロボットでは、両輪の２つの回転センサーを使って、まっすぐに線上を動かしている。このプログラム上の値を変更し、さらにいろいろと動かしてみよう。

```
task main()
{
  SetSensor(IN_1, SENSOR_MODE_ROTATION); ClearSensor(IN_1);
  SetSensor(IN_3, SENSOR_MODE_ROTATION); ClearSensor(IN_3);
  while (true)
  {
    if (SENSOR_1 < SENSOR_3)
    {OnFwd(OUT_A, 75); Float(OUT_C);}
    else if (SENSOR_1 > SENSOR_3)
    {OnFwd(OUT_C, 75); Float(OUT_A);}
    else
    {OnFwd(OUT_AC, 75);}
  }
}
```





このプログラムでは、まず両方のセンサーが回転センサーであることを設定し、値をゼロにリセットしている。つぎからは無限ループとなっている。このループ内では、２つのセンサーが同じ値であればロボットが前進し、一方の回転センサーの値が大きい場合には、両方の値が等しくなるまで一方のモータを減速停止させる。

明らかに、このプログラムは非常に簡単なプログラムとなっている。このプログラムを使って、ロボットを正確な距離まで動かすことや、正確に回転するように拡張してみよう。

ひとつの入力ポートに複数センサー接続（１）

ここで注意事項として、以前（RCXで）は簡単に同じ入力ポートを増設・接続することができていた。しかし、NXTでは６個の増設ケーブル数を超えて拡張することはできなくなっている。

ここでの提案として、唯一確実（かつ簡単）に拡張する方法として、接続ケーブルを使って、タッチ・センサーをアナログによる重ね合せた装置で利用することができる。

もう１つをデジタルによる重ね合わせで、I2CによるNXTで接続して利用する。

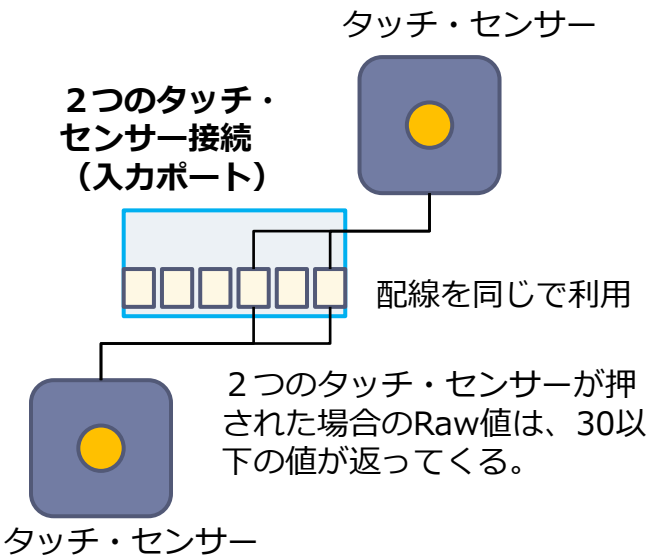
このことは初心者にとっては難しい方法となる。

NXTには、４つの入力ポート数しかない。より複雑（センサーを拡張追加した）なロボットをつくる場合には、この入力ポート数では満足いくものではない。

その他にも特殊な方法で、２つ（さらにもっと）のセンサーを１つの入力ポートに接続することができる。

ひとつの入力ポートに複数センサー接続（１）

最も簡単な方法としては、２つのタッチ・センサーを**同じ入力ポート**に接続することとなる。もし、これらのうちの１つか両方のタッチ・センサーの値は「１」か「０」になる。ただし、どちらのタッチ・センサーが押されたかは区別できない。この区別が必要ではない場合には有効となる。例えば、ロボットの前方と後方にタッチ・センサーを設置し、ロボットがいずれかに動いたときに１つのタッチ・センサーが反応する場合などとなる。しかも、既に述べたような未処理の値（raw value）を設定することもできる。さらに多くの情報を取得することもできる。当然、多くの場合は、両方のタッチ・センサーが同時に反応することはない。２つのタッチ・センサーの違いを、正確に区別することもできる。２つのタッチ・センサーが押された場合の未処理の値は３０よりもさらに低い値が返ってくる。



さらに、タッチ・センサーと光センサーを同じ入力ポートに接続（RCXでは１つ）することができる。この場合、接続タイプを「光センサー」（ただし、光センサーは作動しない）にし、接続モードを未処理の値（raw value）に設定する。この場合、タッチ・センサーが押されたときに値は「１００」以下になります。タッチ・センサーが押されていない時は、光センサーの値が「１００」以下になることはない。このロボットでは、光センサーを下方に向け、前方のバンパーにタッチ・センサーを配置する。両方のセンサーを入力ポート「１」に接続する。このロボットは、明るい状態ではランダムに動き回る。光センサーが暗く反応した場合（未処理の値＞７５０）少し後退する。またタッチ・センサーが何かに触れた場合（未処理の値＜１００）同様に後退する。以下にプログラムを記載する。

ひとつの入力ポートに複数センサー接続（３）

```
mutex moveMutex;
int ttt, tt2;
task moverandom()
{ while (true){
        ttt = Random(500) + 40; tt2 = Random();
        Acquire(moveMutex);
        if (tt2 > 0)
          { OnRev(OUT_A, 75); OnFwd(OUT_C, 75); Wait(ttt); }
        else
          { OnRev(OUT_C, 75); OnFwd(OUT_A,75);Wait(ttt); }
        ttt = Random(1500) + 50;
        OnFwd(OUT_AC, 75);Wait(ttt);
        Release(moveMutex);
    }
}
task submain()
{
     SetSensorType(IN_1, SENSOR_TYPE_LIGHT);
     SetSensorMode(IN_1, SENSOR_MODE_RAW);
     while (true){
          if ((SENSOR_1 < 100) || (SENSOR_1 > 750)){
               Acquire(moveMutex);
               OnRev(OUT_A, 75);Wait(30);
               Release(moveMutex);
          }
     }
}
task main()
{
     Precedes(moverandom, submain);
}
```

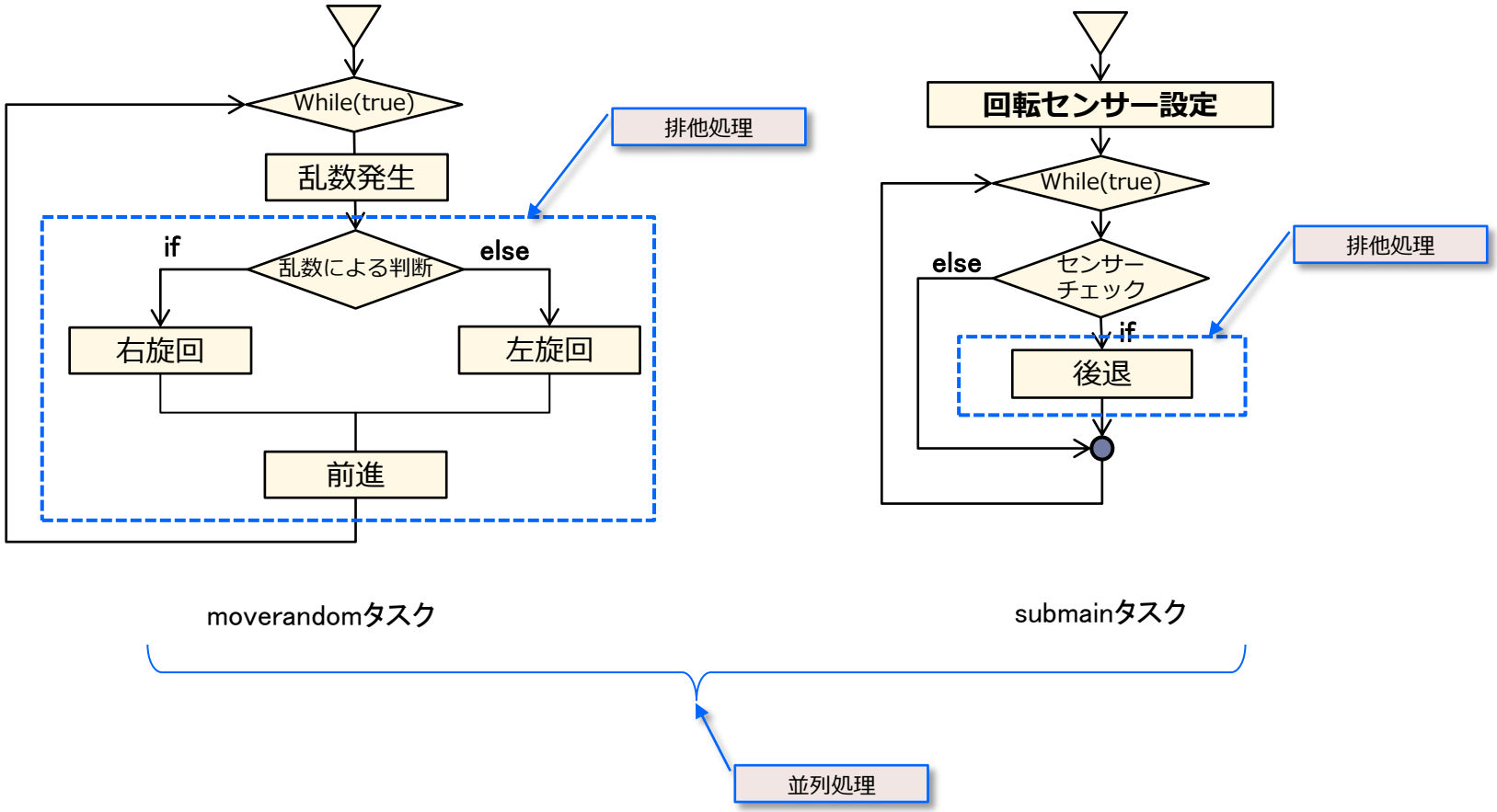
排他処理

排他処理

並列処理

このプログラムは２つのタスクから成り立っている。「moverandom」タスクは、ロボットをランダムに動かす。「main」タスクでは、最初にこの「moverandom」を起動させ、センサーを設定し、その後何が起きるかを待つ。もしセンサーの値が低くなる（タッチ・センサー押された）場合か、高くなる（光センサーが明るいところから外れた）場合には、ランダムに動くのを止め、少し後退し、再度ランダムに動くようになる。

ひとつの入力ポートに複数センサー接続（４）



【展開】

ここでは、センサー関連のAPI 入力関数モジュール群をまとめておく。

モジュール	説明	利用例
SetSensor(port,const config)	センサー設定	SetSensor(S1,SENSOR_TOUCH);
SetSensorType(port,type)	センタータイプ設定	SetSensorType(S1,SENSOR_TYPE_TOUCH);
SetSensorMode(port,mode)	センサーモード設定	SetSensorMode(S 1 ,SENSOR_MODE_RAW);
SetSensorLight(port)	光センサー設定	SetSensorLight(S1);
SetSensorSound(port)	サウンドセンサー設定	SetSensorSound(S1);
SetSensorTouch(port)	タッチセンサー設定	SetSensorTouch(S1);
SetSensorLowspeed(port)	超音波センサー設定	SetSensorLowspeed(S1);
SetSensorColorBlue(port)	青色発光カラーセンサー設定	SetSensorColorBlue(S1);
SetSensorColorFull(port)	フル発光カラーセンサー設定	SetSensorColorFull(S1);
SetSensorColorGreen(port)	緑色発光カラーセンサー設定	SetSensorColorGreen(S1);
SetSensorColorNone(port)	光なしカラーセンサー設定	SetSensorColorNone(S1);
SetSensorColorRed(port)	赤色発光カラーセンサー設定	SetSensorColorRed(S1);
ResetSensor(port)	センサーの値をリセット	ResetSensor(S1);

まとめ

- この章ではセンサーについて多くの詳しい内容を学んだ。
- センサーの接続タイプと接続モードをそれぞれどう設定するか、それにその他の情報をどう利用するかを見てきた。
- また回転センサーの使い方についても学んだ。
- さらにどう２つのセンサーを重複させてNXTの１つの入力ポートに接続するかも見てきた。
- これらの秘訣は、もっと複雑なロボットを組み立てるとき、とても便利なものとなる。
- センサーは、常時これらの重要な役割に活用できる。

１０．並列タスク

前に指摘したように、NXCのタスクは、同時もしくは並列で、実行することができる。このことは、非常に便利なもので、１つのタスクでセンサーを見ながら、他のタスクでロボットを動かし、さらに他のタスクで音楽をプレイすることができる。しかし並列処理は、問題発生の原因にもなる。たとえば、あるタスクが他のタスクを妨害することなど。ここでは、並列処理についての注意点などを学ぶこととする。

覚える事

1．間違いプログラム
2．並列処理
3．排他処理
4．割込み処理
5．Mutex処理
6．セマフォ―処理
6．危険領域

間違ったプログラム（１）

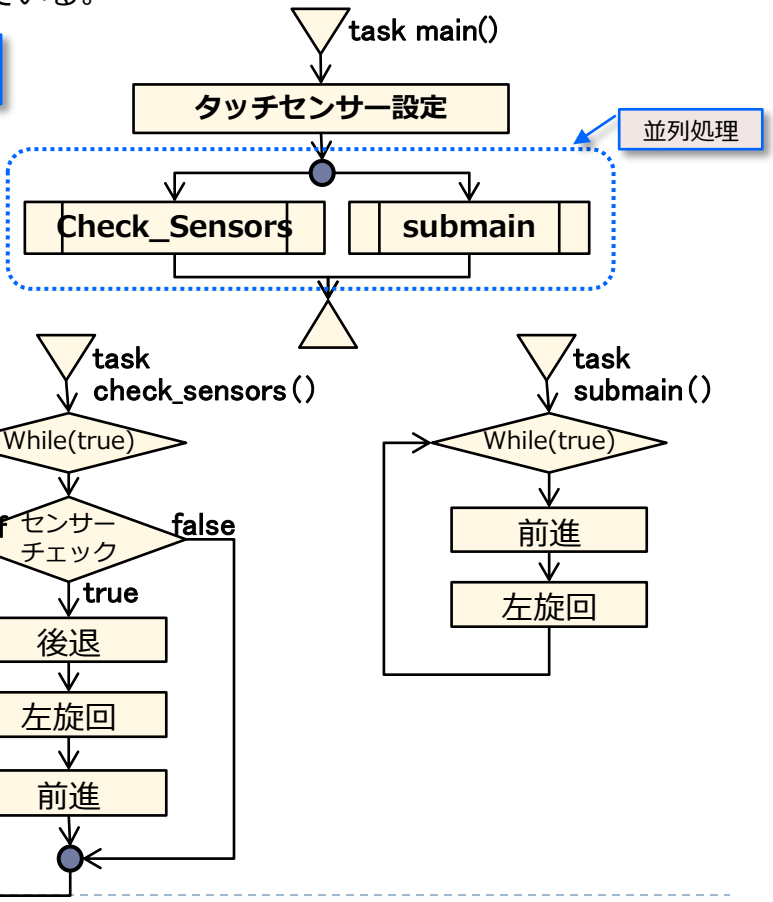
```
task check_sensors()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      OnRev(OUT_AC, 75);  Wait(500);
      OnFwd(OUT_A, 75); Wait(850);
      OnFwd(OUT_C, 75);
    }
  }
}

task submain()
{
  while (true)
  {
    OnFwd(OUT_AC, 75); Wait(1000);
    OnRev(OUT_C, 75); Wait(500);
  }
}

task main()
{
 SetSensor(IN_1, SENSOR_TOUCH);
 Precedes(check_sensors, submain);
}
```

センサー反応で
後退→左旋回→前進

常に繰り返し
前進→左旋回

並列処理

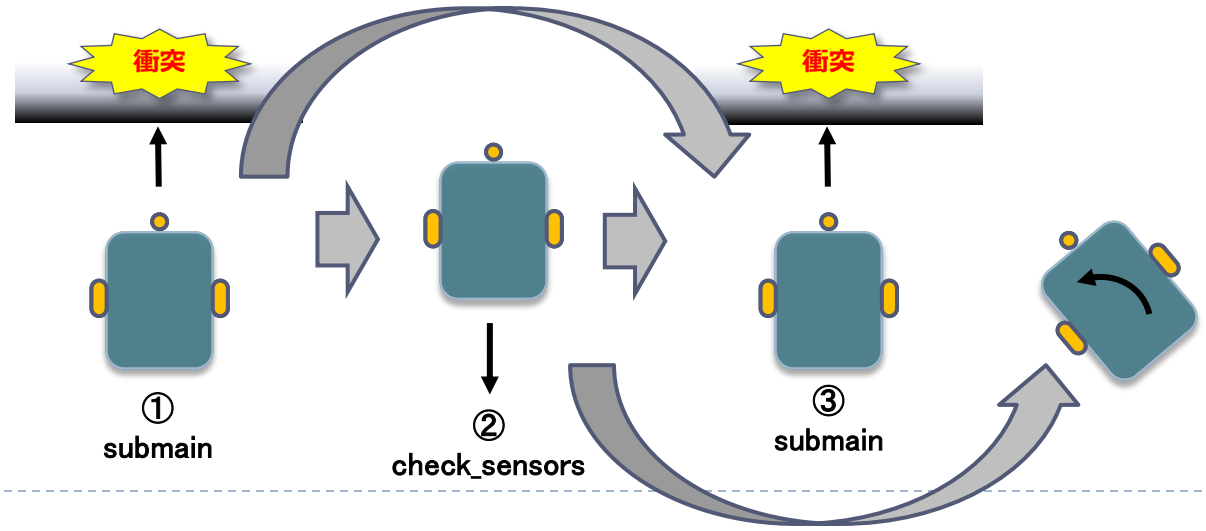
つぎのプログラムを見てみよう。タスク**submain**では、ロボットが四角形を描くように動き（以前記述したプログラム）、タスク**check_sensors**ではタッチ・センサーをチェックしている。タッチ・センサーが何かに触れたとき、少し後退し、９０度回転させている。



間違ったプログラム（２）

このプログラムでは、問題は無いように見える。しかし、実行させてみると、必ずしも期待したような動きにならないことに気づくであろう。

つぎのことを実行してみよう。ロボットが回転しているときに、① 何かでタッチ・センサーに触れてみよう。
すると、② 後ろに動きだし、③ 瞬時に再び前進し、障害物に衝突する。
この理由は、②で「**check_sensor**」が起動しはじめたが、再度③の「**submain**」が起動し妨害したためとなる。
ロボットが「**submain**」で起動しはじめた時、「**check_sensors**」タスクは待機状態に入っている。つぎに、タッチ・センサーに何かが衝突したとき、この最初の「**check_sensors**」タスクが起動し、ロボットは後退しはじめ、その一瞬で、「**submain**」タスクが継続状態であったことで、再びロボットを障害物の方向へ前進させるようになる。
再度衝突しても「**check_sensors**」タスクは気付かない状態となる。このプログラムでは、期待した動きをするかどうかは、明確にすることは難しい。
この問題は、「**submain**」タスクが継続している間、最初の「**check_sensors**」タスクが動いている状態にあるからで、しかも２つめの「**submain**」タスクの実行を妨害するようになっているからである。



互いのタスクが起動しても、無視して継続状態になる。プログラムの衝突（コンフリクト）と呼ばれる

危険領域と mutex変数（１）

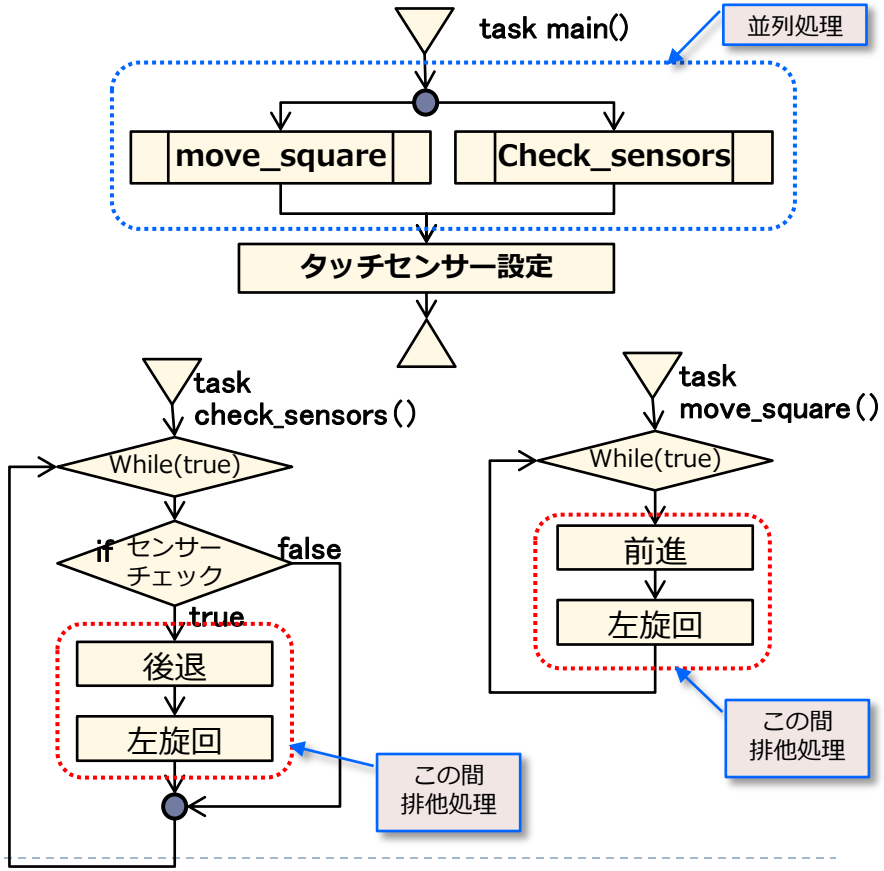
```
mutex moveMutex;

task move_square()
{
  while (true)
  {
    Acquire(moveMutex);
    OnFwd(OUT_AC, 75); Wait(1000);
    OnRev(OUT_C, 75); Wait(500);
    Release(moveMutex);
  }
}

task check_sensors()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      Acquire(moveMutex);
      OnRev(OUT_AC, 75); Wait(500);
      OnFwd(OUT_A, 75); Wait(500);
      Release(moveMutex);
    }
  }
}

task main()
{
   Precedes(move_square, check_sensors);
   SetSensorTouch(IN_1);
}
```

排他処理

排他処理

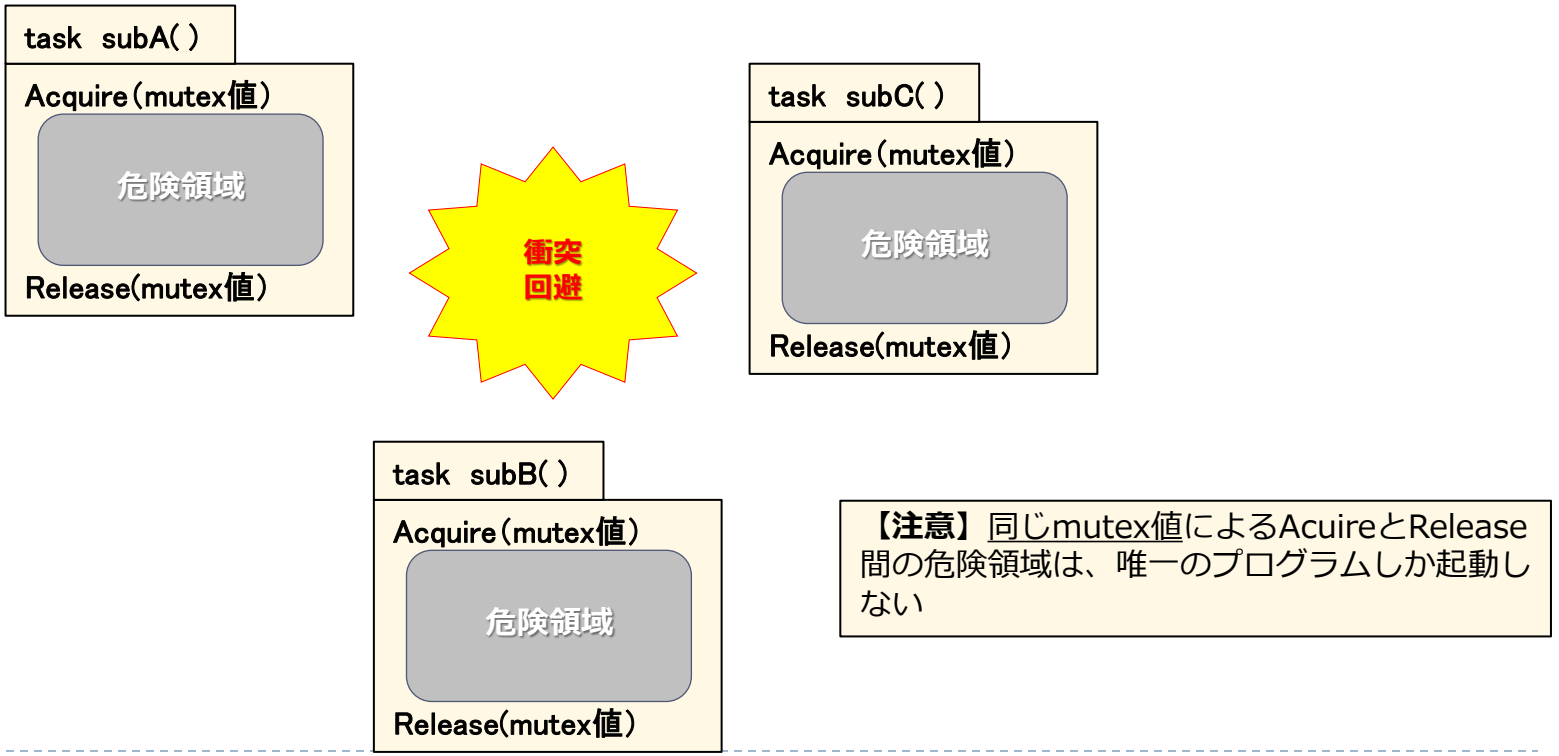
上記の問題を避ける１つの方法として、どんな状態でも１つのタスクでロボットを動かしていることを確認するようにすることである。

つまり、このことはすでに６章で紹介している「**排他処理**」を使うことになる。



危険領域と mutex変数（２）

ここでのポイントは、「**check_sensors**」と「**move_square**」の２つのタスクは、それぞれ他のタスクが稼動していない状態で、モータを制御することができる。プログラム上では、モータを制御する前に、相互排除の値「**moveMutex**」を使って「**Acquire**」ステートメントで宣言し、他のタスクを待機状態にする。
この「**Acquire**」コマンドの対となる「**Release**」コマンドは、「mutex」値をフリーにし、他のタスクでモータを制御することができるようになる。
この「**Acquire**」と「**Release**」との間を「**危険領域（critical region）**」と呼び、プログラムが単独で動いていることを意味します。この方法によって、互いのタスク間の衝突を避けることができることとなる。



【注意】同じmutex値によるAcuireとRelease間の危険領域は、唯一のプログラムしか起動しない

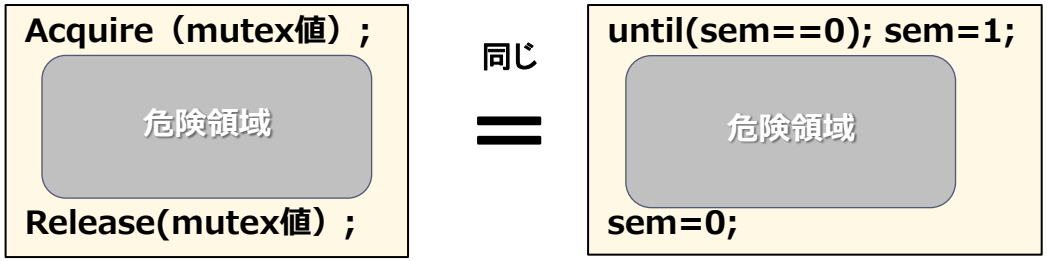
セマフォーの利用（１）

ここで紹介している「**Acquire**」と「**Release**」コマンドで使っている「**mutex**」値の代わりとなるプログラミングを紹介する。

プログラムの衝突を避けるための標準的な標準的な問題解決手法としては、タスク内でモータ制御状態を示す変数値を使うことである。

他のタスクでモータ制御ができないように、この変数値を利用する。この変数値を「**セマフォー（semaphore：シグナル）**」と呼び、「sem」といったようなセマフォー（mutexと同じような）を使う。この場合の「sem」の値がゼロ（０）の場合は、モータを利用しているタスクはないことを意味する。

２つの排他処理方法

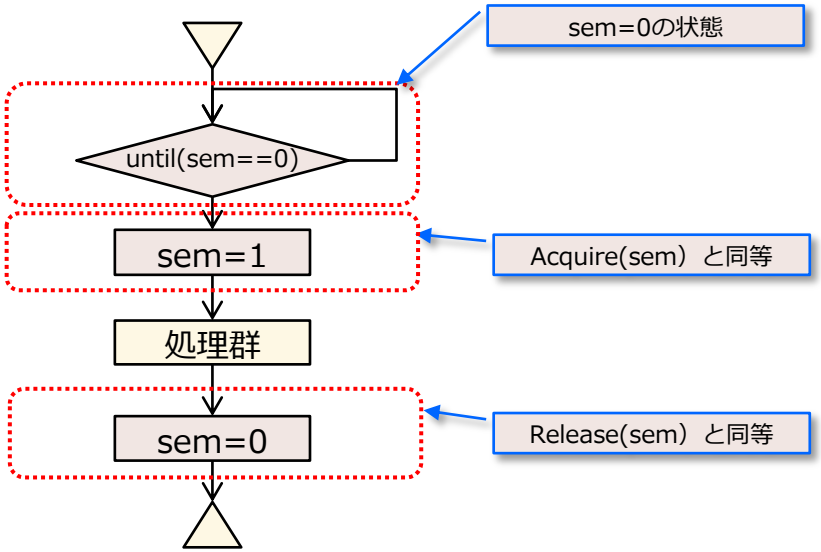


mutex による方法　　**セマフォーによる方法**

セマフォーの利用（２）

それでは、タスクでモータを動かしている状態を示す場合について、つぎに示してみよう。

```
until (sem == 0);
sem = 1;   //Acquire(sem);
// Do something with the motors「モータ制御」
// critical region 「重要（排他）領域」
sem = 0;   //Release(sem);
```



このプログラムでは、モータが使われなくなる「sem= 0 」になるまで待機し、その後に排他制御として「sem」に「１」を設定する。
この状態でモータを制御することができるようになる。
処理が終わったら、また「sem」を「０」に戻す。
以上のプログラムで、このセマフォーを使った処理が実現できることになる。
例えば、タッチ・センサーが何かに接触したとき、セマフォーが設定され、バックアップされている処理が動き始める。この処理を行っている間、「move_square」タスクは待機状態にある。つまりバックアップされた状態で、セマフォーは「０」に設定されたまま「move_square」は継続される。

【注意】このセマフォーは、シグナルをOnにしたりOffにしたりした状態と考えられ、Onの場合は安全にタスクを起動するが、Offの場合には待機状態となる。

セマフォーの利用（３）

```
int sem;
task move_square()
{
  while (true)  {
    until (sem == 0); sem = 1;
    OnFwd(OUT_AC, 75);
    sem = 0;
    Wait(1000);
    until (sem == 0); sem = 1;
    OnRev(OUT_C, 75);
    sem = 0;
    Wait(850);
  }
}
task submain()
{
  SetSensor(IN_1, SENSOR_TOUCH);
  while (true)
  {
    if (SENSOR_1 == 1)  {
      until (sem == 0); sem = 1;
      OnRev(OUT_AC, 75); Wait(500);
      OnFwd(OUT_A, 75); Wait(850);
      sem = 0;
    }
  }
}
task main()
{
  sem = 0;
  Precedes(move_square, submain);
}
```

【注意】グローバル変数宣言

【注意】初期設定

この「move_square」内で、セマフォーを「１」にし、さらに「０」に戻すことに着目してみよう。これは便利なもので、２つのコマンドで「OnFwd()」を使って処理をしている（８章参照）。他のタスクでこのコマンド処理を中断させる必要はなくなる。
セマフォーは、とても使い勝手がよく、並列処理での複雑なプログラムを作成する場合にもほとんど利用できる。

【注意】ここでセマフォーを利用する注意点として、
（１） グローバル変数として宣言すること
（２） 初期値を設定すること

【展開】

■「**safecall**」キーワード
もし、関数が「safecall」を使っていた場合、コンパイラは「Acquire」と「Release」の中で呼ばれた関数と同じように、クロスした並列なスレッドで、この関数は同期をとって実行することになる。つまり、「safecall」が付いた関数は、自分自身が複数呼ばれた場合には、先に実行されているものがあれば、つぎの実行は待機状態となる。

```
safecall void foo(unsigned int frequency)
{
   PlayTone(frequency, SEC_1);
   Wait(SEC_1);
}

task task1()
{
  while(true) {
    foo(TONE_A4);
    Yield();
  }
}
task task2()
{
  while(true) {
    foo(TONE_A5);
    Yield();
  }
}
task main()
{
   Precedes(task1, task2);
}
```

まとめ

- この章では、異なるタスクを使う場合に、**問題が発生**するプログラムについて学んだ。
- その結果は、いつも注意する必要がある。期待しない動きがよくある。この問題を解決するための２つの手法を学習した。
- 最初の解決法が、ひとつのタスク内において、常に排他処理を制御するように**タスクの中断と再起動を行う方法**。
- つぎの解決法は、各タスク内の実施において、**セマフォーを使って制御する方法**。
- これらを学んだことで、並列処理において、いつの瞬間でも一つのタスクを実行させることができるようになった。

１１．ロボット間の通信

この章では、二つ以上のNXT を持っている場合に役立つものを紹介する。

（またPCを使って１つのNXTとのデータ通信も含む）

NXTのロボットは、**Bluetooth**（無線技術）を使って通信ができ、複数のロボット間で協業（もしくは戦闘）させることもでき、さらに２つのNXTを使って６つのモータと８つのセンサーを装備した複雑なロボットを構築することもできる。

古いRCXでは、「**InfraRed**」メッセージを送って、すべてのロボットでこれを受信することができる。

NXTは、特殊な通信方法を採用していて、本体の「Bluetooth」メニューで、２つ以上のNXT（もしくはPC）と接続し、メッセージを送ることができるようになっている。

最初に繋いだNXTを**マスター**（**master**：主人）と呼び、他を**スレイブ**（**slave**：奴隷）と呼び、３つのスレイブを、マスター側の３つの「ライン１」「ライン２」「ライン３」のBluetoothに接続する。

スレイブ側は、常に接続しているマスターの「ライン０」を見ている状態となる。

メッセージとして最大１０個の**メールボックス**を送信することができる。

覚える事

１．Bluetooth通信
２．マスターとスレイブ
３．メールボックス
４．メッセージ送信
５．送信数の承認

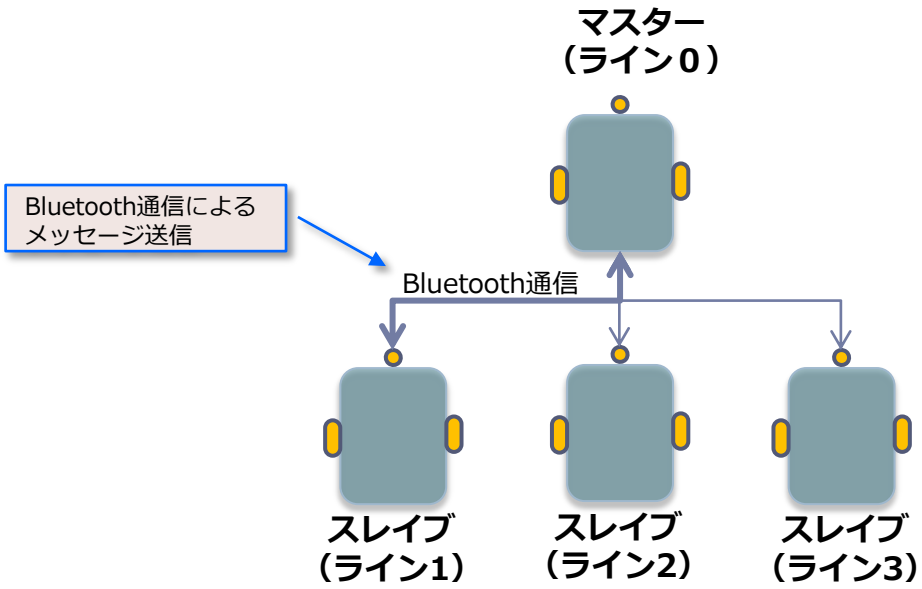
【注意】ここで利用する複数のNXTどうしは、あらかじめマスター側で、スレイブの認識を行う必要がある。
ステップ１：それぞれ電源をOnにし、NXTデスプレィ上のコマンドで、Bluetooth機能をOnの状態にしておく。
ステップ２：つぎにマスター側のNXTディスプレィ上で、スレイブとのBluetooth通信割り当てで、ライン１からライン3の何れかを選んで、スレイブと接続する。

マスターとスレイブ間のメッセージ送信（１）

ここでは **マスター**側と**スレイブ**側との**メッセージ送信**について紹介する。

基本的なプログラムでは、**Bluetooth**による無線ネットワーク上の２つのNXTが文字列のメッセージを高速に通信しあう。

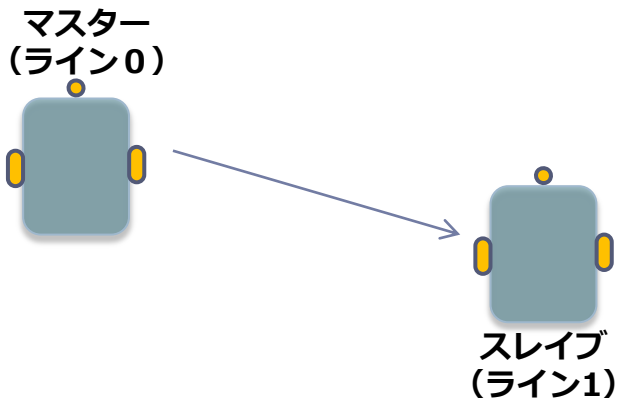
マスター側のつぎのプログラムでは、最初に「**BluetoothStatus(conn)**」機能を使い、正しくスレイブが「**ライン１**（conn=1)」に接続されているかどうかをチェックし、「M」文字を使って通信構築を行ってメッセージを送信し、増数の「**SendRemoteString (conn,queue,string)**」を使い、さらにメッセージがスレイブ側から「**ReciveRemoteString(queue,crear,string)**」を通じて受信し、そのデータが画面に表示されるようになっている。



マスターとスレイブ間のメッセージ送信（２）

```
//MASTER
#define BT_CONN 1
#define INBOX 1
#define OUTBOX 5

sub BTCheck(int conn){
  if (!BluetoothStatus(conn) == NO_ERR){
    TextOut(5, LCD_LINE2, "Error");
    Wait(1000);
    Stop(true);
  }
}
task main(){
  string in, out, iStr;
  int i = 0;
  BTCheck(BT_CONN); //スレイブの接続をチェックする
  while(true){
    iStr = NumToStr(i);
    out = StrCat("M",iStr);
    TextOut(10, LCD_LINE1, "Master Test");
    TextOut(0, LCD_LINE2, "IN:");
    TextOut(0, LCD_LINE4, "OUT:");
    ReceiveRemoteString(INBOX, true, in);
    SendRemoteString(BT_CONN, OUTBOX, out);
    TextOut(10, LCD_LINE3, in);
    TextOut(10, LCD_LINE5, out);
    Wait(100);
    i++;
  }
}
```

マスター側でのBluetooth通信によるスレーブ【conn】のメッセージ送信チェック
<エラーの場合の処理>

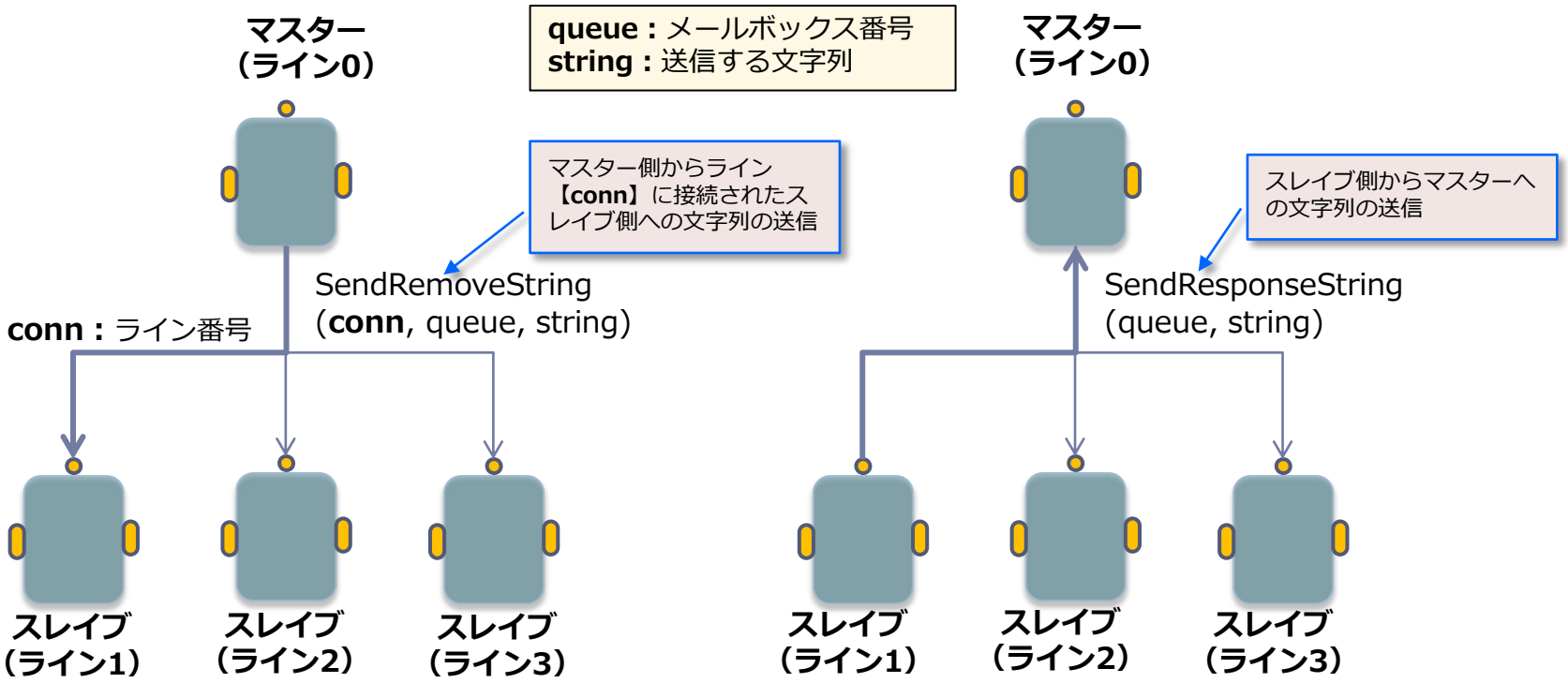


スレイブ側が正しく接続されているかをチェックし、未接続の場合には中止。

接続の場合には、ディスプレイ上に、スレイブから送られたメッセージ表示と、マスターが送った「M0」を表示する。

マスターとスレイブ間のメッセージ送信（３）

マスターのプログラムで使われている「**SendRemoveString(conn, queue, string)**」の代わりに、スレイブ側も似た様なプログラムとして、「**SendResponseString(queue, string)**」を使っているが、スレイブ側は「ライン０」を使ってマスター側にメッセージを送るだけとなることから、connのパラメータは不要となる。



マスターとスレイブ間のメッセージ送信（４）

```
// SLAVE
#define BT_CONN 1
#define INBOX 5
#define OUTBOX 1

sub BTCheck(int conn){
  if (!BluetoothStatus(conn) == NO_ERR){
    TextOut(5, LCD_LINE2, "Error");
    Wait(1000);
    Stop(true);
  }
}
task main(){
  string in, out, iStr;
  int i = 0;
  BTCheck(0); //マスターの接続をチェックする
  while(true){
    iStr = NumToStr(i);
    out = StrCat("S",iStr);
    TextOut(10, LCD_LINE1, "Slave Test");
    TextOut(0, LCD_LINE2, "IN:");
    TextOut(0, LCD_LINE4, "OUT:");
    ReceiveRemoteString(INBOX, true, in);
    SendResponseString(OUTBOX, out);
    TextOut(10, LCD_LINE3, in);
    TextOut(10, LCD_LINE5, out);
    Wait(100);
    i++;
  }
}
```

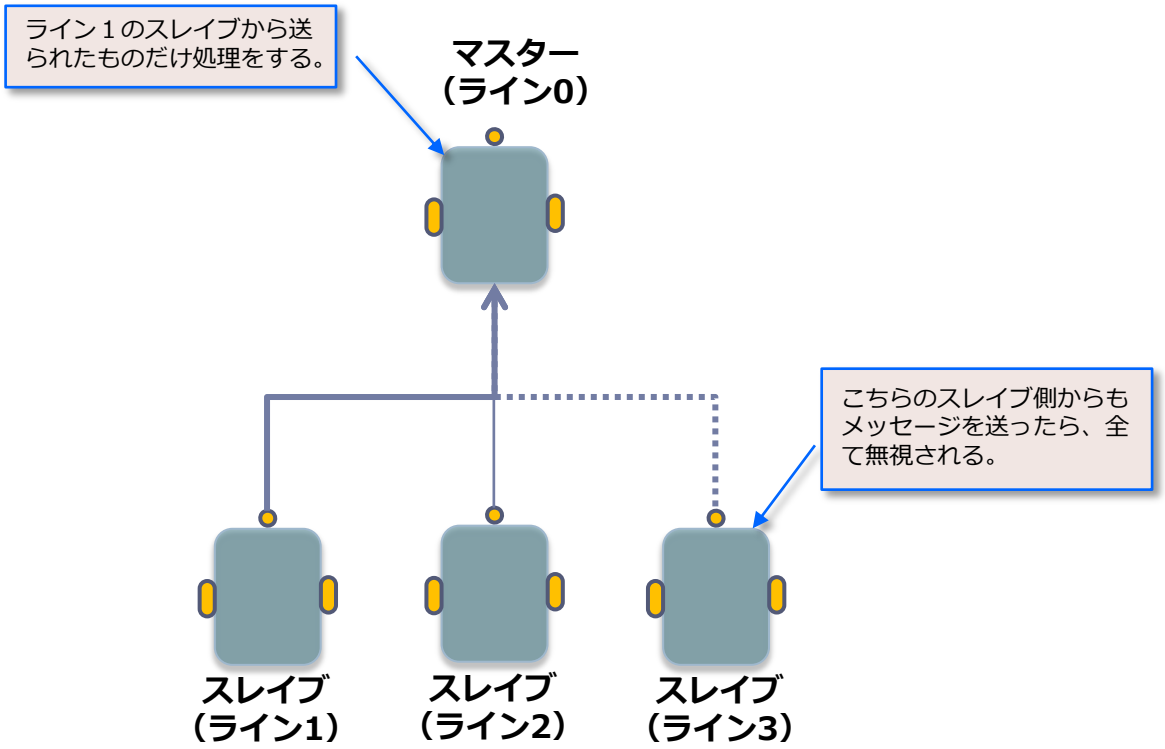
この部分のサブルーチンは、マスター側と同じことに注意。Mainタスクでの引数で使い分けをしている。

このプログラムで、ひとつ問題があることに気づくだろう。他のスレイブがメッセージを送り続けたとき、すべての送信されたメッセージが無くなっていることを知らずに、表示されている数値は増えつづける。このことは、マスター側が他に受信できないことを意味する。この障害を避けるために、正しいプロトコルのための配信承認について学んでおく必要がある。

マスターとスレイブ間のメッセージ送信（５）

前述のプログラムで、ひとつ問題があることに気づくだろう。他のスレイブがメッセージを送り続けたとき、すべての送信されたメッセージが無くなっていることを知らずに、表示されている数値は増えつづける。このことは、マスター側が他に受信できないことを意味する。この障害を避けるために、正しいプロトコルのための**配信承認**について学んでおく必要がある。

前述プログラムの問題点



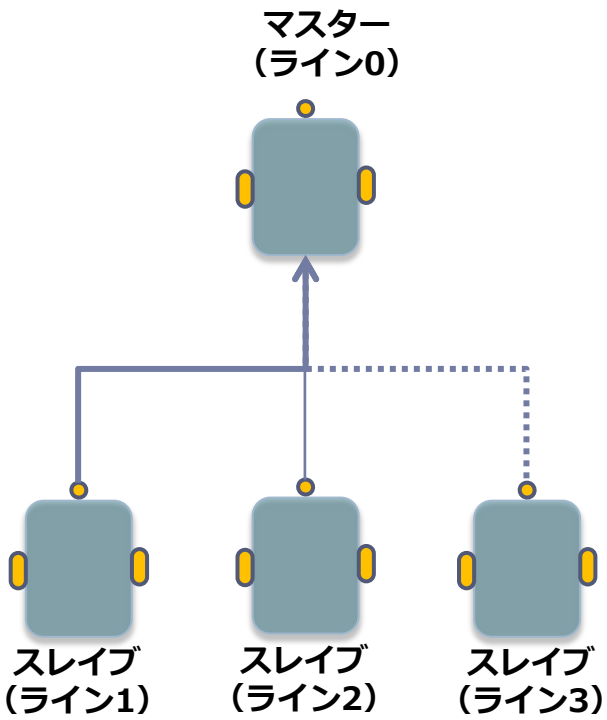
送信数の承認【ack】（１）

ここで他に２つのプログラムを見てみよう。

マスターは、このとき「**SendRemoteNumbaer(conn, queue, number)**」を使って数字を送信し、スレイブ側の承認のための待機を中断し（「until」サイクルの中で「**ReceiveRemoteString**」を使用）、スレイブが受信中で承認を送信するだけとなり、マスターはつぎのメッセージを送信し続けることとなる。スレイブは、単純に「**ReceiveRemoteNumber(queue, clear, number)**」を通じて数値を受信し、「**SendResponseNumber**」を通じて承認を送信する。

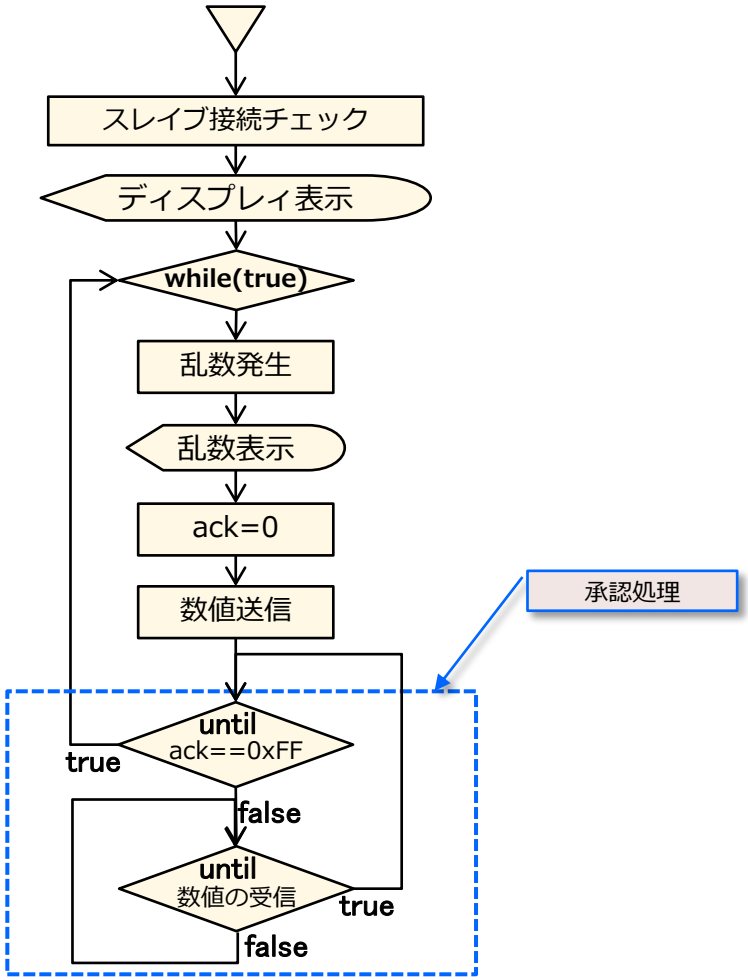
このマスターとスレイブの両方のプログラムでは、承認のための**共通コード**（この中では16進数の「**0xFF**」）を使って合意している。

このプログラムでは、マスターは、乱数の数値を送信し、スレイブ側から正しいコードが返ってくることで承認を待ち続けるが、一方でマスターは、新しい承認なしで送信を続けている。理由は、ここでの値が変わっていくからとなる。



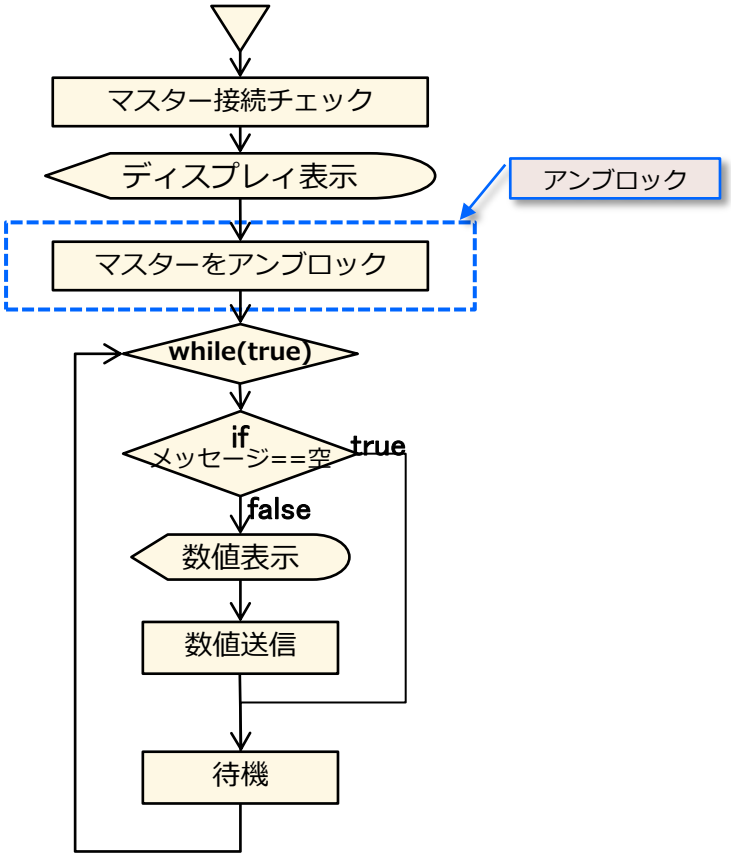
送信数の承認【ack】（２）

```
//MASTER
#define BT_CONN 1
#define INBOX 1
#define OUTBOX 5
#define CLEARLINE(L) TextOut(0,L, " ")
sub BTCheck(int conn){
  if (!BluetoothStatus(conn) == NO_ERR){
    TextOut(5, LCD_LINE2, "Error");
    Wait(1000);
    Stop(true);
  }
}
task main(){
  int ack;
  int i;
  BTCheck(BT_CONN); //スレイブの接続をチェックする
  TextOut(10, LCD_LINE1, "Master Sending");
  while(true) {
    i = Random(512);
    CLEARLINE(LCD_LINE3);
    NumOut(5, LCD_LINE3, i);
    ack = 0;
    SendRemoteNumber(BT_CONN, OUTBOX, i);
    until(ack == 0xFF){
      until(ReceiveRemoteNumber(INBOX, true, ack)
        == NO_ERR);
    }
    Wait(250);
  }
}
```



送信数の承認【ack】（３）

```
// SLAVE
#define BT_CONN 1
#define IN_MBOX 5
#define OUT_MBOX 1

sub BTCheck(int conn){
  if (!BluetoothStatus(conn) == NO_ERR){
    TextOut(5, LCD_LINE2, "Error");
    Wait(1000);
    Stop(true);
  }
}
task main(){
  int in;
  BTCheck(0); //マスターの接続をチェックする
  TextOut(5, LCD_LINE1, "Slave Receiving");
  SendResponseNumber(OUT_MBOX, 0xFF);
                    //マスターをアンブロックする
  while(true){
    if(ReceiveRemoteNumber(IN_MBOX, true, in) !=
          STAT_MSG_EMPTY_MAILBOX){
      TextOut(10, LCD_LINE3, " ");
      NumOut(5, LCD_LINE3, in);
      SendResponseNumber(OUT_MBOX, 0xFF);
    }
    Wait(10);//オプションの待機時間
  }
}
```

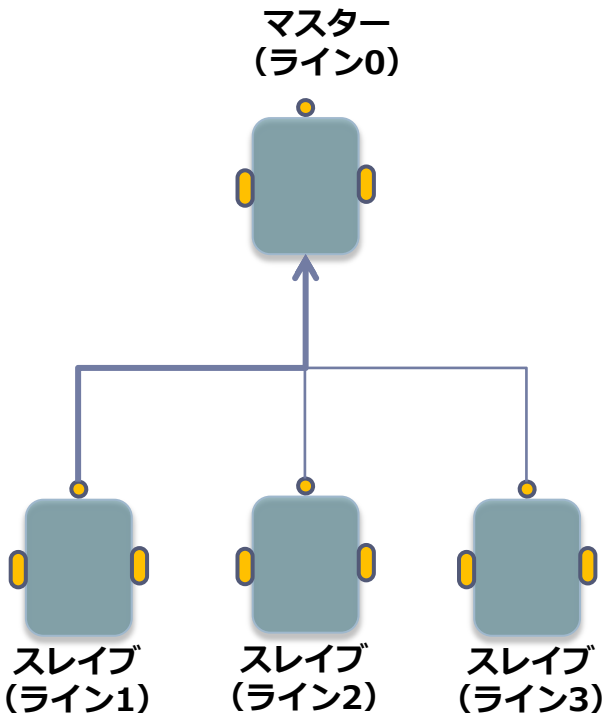


送信数の承認【ack】（４）

スレイブは**メールボックス**を常にチェックし、空でない場合にその値を読んでNXT画面に表示し、マスターに**承認**を返します。

プログラムの最初に、**メッセージ**を読み取ることなしにマスターに承認を送信しています。この秘策なしでは、マスターが最初に走り出し、スレイブが後でスタートした場合に**ハングアップ**（エラー発生）します。

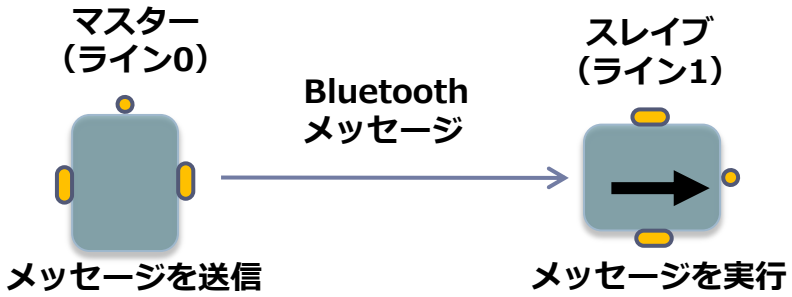
この方法では、最初のメッセージのいくつかは失われますが、マスターとスレイブのプログラムをハングアップのリスクなしに、異なる時間でスタートさせることができます。



直接的なコマンド（１）

Bluetooth通信機能には他にもいろいろな機能が用意されている。この中には、マスターから直接スレイブを制御できる機能がある。

つぎの例では、マスターからスレイブへ直接コマンドを送り、音を出してモータを動かしている。この場合には、スレイブ側のプログラムは不要となる。この場合は、スレイブ側のNXTは、受信したメッセージをファームウェアを通して実施する。



マスターから直接スレイブを動かす方法

直接的なコマンド（２）

```
//MASTER
#define BT_CONN 1
#define MOTOR(p, s) RemoteSetOutputState(BT_CONN, p, s, ¥
  OUT_MODE_MOTORON+OUT_MODE_BRAKE+OUT_MODE_REGULATED, ¥
  OUT_REGMODE_SPEED, 0, OUT_RUNSTATE_RUNNING, 0)

sub BTCheck(int conn){
  if (!BluetoothStatus(conn) == NO_ERR){
     TextOut(5, LCD_LINE2, "Error");
     Wait(1000);
     Stop(true);
   }
}
task main(){
   BTCheck(BT_CONN); //スレイブの接続をチェックする
   RemotePlayTone(BT_CONN, 4000, 100);
   until( BluetoothStatus(BT_CONN) == NO_ERR);
   Wait(110);
   RemotePlaySoundFile(BT_CONN, “! Click.rso”, false);
   until( BluetoothStatus(BT_CONN) == NO_ERR);
   // Wait(500);
   RemoteResetMotorPosition(BT_CONN, OUT_A, true);
   until( BluetoothStatus(BT_CONN) == NO_ERR);
   MOTOR(OUT_AC, 100);
   Wait(1000);
   MOTOR(OUT_AC, 0);
}
```

MOTORコマンド宣言
もとは、RemoteSetOutPutStateコマンド

Bluetoothで、もう1つの
NXTの接続可否をチェック

リモートで、他のNXT
上で音を発生

リモートで、他のNXT
上で音楽を再生

リモートで、他のNXT
上で音楽を再生

#define で定義した
MOTORで前進

【展開】

以下のAPI関数は、リモートで、複数のNXTどうしのBluetoothを使って処理するものとなる。

モジュール	引　数	説明
RemoteMessageRead	conn,que	リモートによるメッセージの読込み
RemoteMessageWrite	conn,que,msg	リモートによるメッセージの書込み
RemoteStartProgram	conn,fname	リモートによるプログラムの起動
RemoteStopProgram	conn	リモートによるプログラムの停止
RemoteSetOutputState	conn,po,sp,md,tp,rst,tlim	リモートによる出力関連設定
RemotePlayTone	conn,freq,du	リモートによる音のプレイ
RemotePlaySoundFile	conn,fname,bloop	リモートによるサウンドファイルのプレイ
RemoteResetMotorPosition	conn,po,bR	リモートによるモータ駆動

詳細は、NXCのヘルプ画面参照

まとめ

- この章では、２つ以上のロボット間でBluetooth通信を使って、文字列や数値の送信受信や、通信承認の待機などの接続について学んだ。
- 最後の事項は、安全な通信プロトコルが必要なときに重要なものとなる。
- その他に直接コマンドを使ってスレイブを動かすことも学んだ。
- 複数のNXTを使った、高度なプログラミングができることを学んだ。

１２．その他のコマンド

NXC は、以上のほかにも多くのコマンドを用意している。
この章では３つのタイプのコマンド、① タイマーの使い方、
② 画面表示のコマンド、それと ③ NXTファイルシステムの
使用法について紹介している。

覚える事

１．タイマー機能
２．画面表示
３．文字表示
４．グラフィック表示
５．描画ファイル出力
６．ふらっしゅめもり
７．ファイル入出力
８．プリプロセッサ

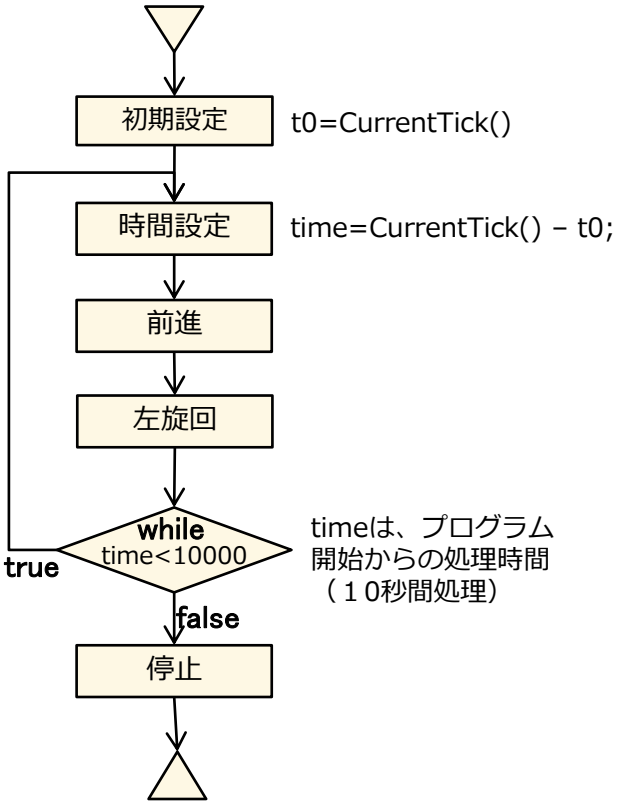
タイマーの使い方（１）

NXTでは、実行を継続させるタイマー機能がある。このタイマーは、1000分の1秒（ミリ秒）毎の刻みとなっていて、現時点の時刻は、「**CurrentTick**()」を使って値を取り出す。以下にタイマーを使った事例を紹介する。

つぎのプログラムでは、１０秒間でランダムな数値を使ってロボットを動かしている。

ここでのプログラムは、既に４章で紹介したプログラムと比較してみよう。このタイマーは、簡単で正確な機能となっている。

```
task main()    //  Program12.1
{
  long t0, time;
  t0 = CurrentTick();
  do
  {
    time = CurrentTick() – t0;
    OnFwd(OUT_AC, 75);
    Wait(Random(1000));
    OnRev(OUT_C, 75);
    Wait(Random(1000));
  }
  while (time < 10000);
  Off(OUT_AC);
}
```

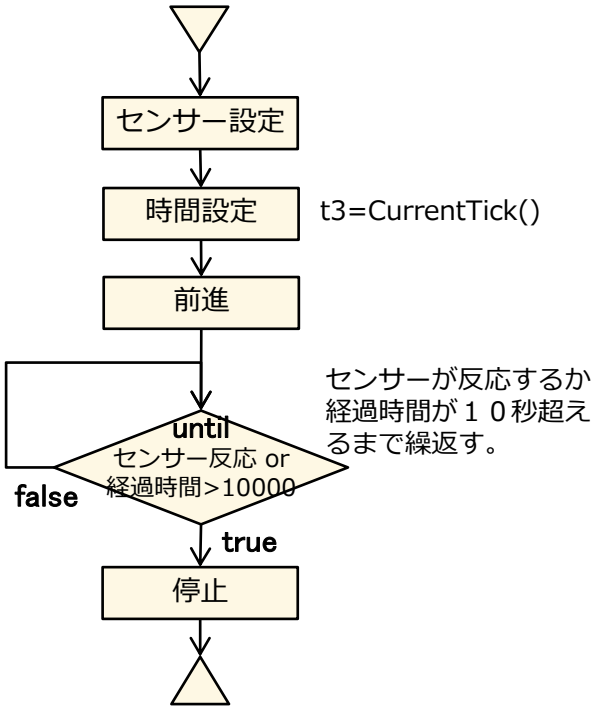


【注意】タイマーの単位は、「Wait」コマンドと同様に、1000分の１（1/1000） 秒（ミリ秒）であることに注意してください。

タイマーの使い方（２）

これまで使ってきた「Wait()」コマンドを、このタイマーを使って置き換えることは非常に便利となる。
タイマーをリセットし、つぎの特別な値になるまで待機するように正確な総合時間で停止させることができる。しかも、この待機中に、別なイベント（例えば、センサーなど）で反応せることもできる。
つぎの簡単なプログラムで、ロボットを１０秒経過するまでか、タッチ・センサーが反応するまで前進させる。

```
task main()    //  Program12.2
{
  long t3;
  SetSensor(IN_1,SENSOR_TOUCH);
  t3 = CurrentTick();
  OnFwd(OUT_AC, 75);
  until ((SENSOR_1 == 1) || ( (CurrentTick() – t3) > 10000) );
  Off(OUT_AC);
}
```



【注意】この中で、「|||」の表記は、「論理OR」の意味がある。
他に「==」（論理等）、「!=」（論理不等）、「&&」（論理AND）

タイマーの使い方（３）

以上で紹介した時間制御に関するタスクや関数を紹介しておく。

モジュール	説　明	事　例
Wait(ms)	ミリ秒での待機時間設定（ミリ秒）	Wait(x);
CurrentTick()	現時刻（ミリ秒）	x = CurrentTick();
FirstTick()	プログラム起動時刻（ミリ秒）	x = FirstTick();
SleepTime()	自動シャットダウン時刻	x = SleepTime();
SleepTimer()	自動シャットダウンからの時刻	x = SleepTimer();
ResetSleepTimer()	前回自動シャットダウン時刻のリセット	ResetSleepTimer();
SetSleepTime(min)	待機時間の設定（秒）	SetSleepTime(8);
SetSleepTimer(min)	待機時間の設定（秒）	SetSleepTimer(4);
Stop(vl)	vl値が真でストップ	Stop(x==30);
StopAllTasks()	現在実行中のタスク全てを中断	StopAllTasks();
StopTask(task)	タスクを中断	StopTask(sound);
StartTask(task)	タスクを起動	SartTask(movement);
Acquire(mutex)	Mutex値による他のタスクを中断	Acquire(mMutex);
Release(mutex)	Mutex値による他のタスクを再開	Release(mMutex);
Precedes(task1,task2,･･･)	タスク群の並列処理	Precedes(sub1,sub2,sub3);
Follow(taks1,task2,･･･)	タスクを順に処理	Follows(sub1,sub2);
ExitTo(task)	現在のタスクを中断し、次taskに切り替え	NextTo(nextTask);

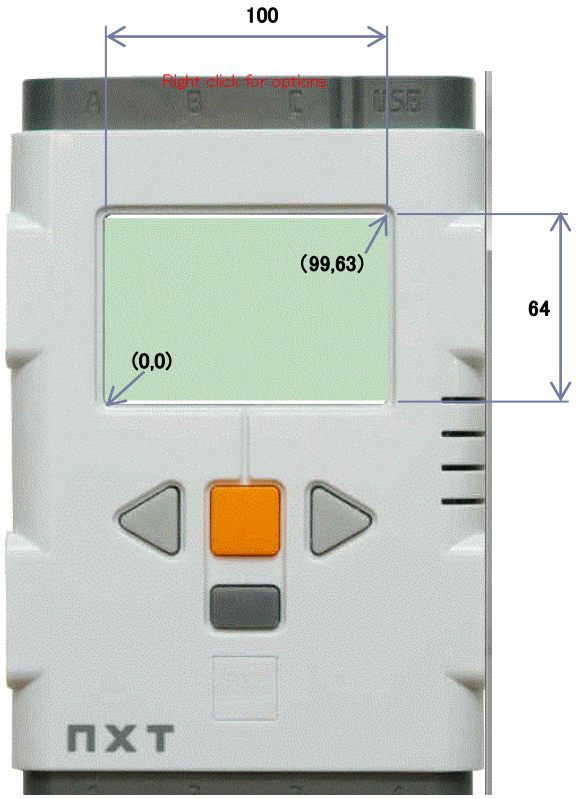
詳細は、NXCのヘルプ画面参照

ドットマトリックス・ディスプレイ（１）

NXT は**ドット・マトリクス表示**機能を持っていて、横100ピクセル、縦64ピクセルのモノクロのディスプレイとなる。NXCでは、テキスト文字、数字、ドット、線分、長方形、円、それにビットマップイメージ(「**.ric**」ファイル)を描画・表示させる多くのAPI機能がある。ここでのピクセル座標値（０，０）は、ディスプレイ上の右上を意味する。

【注意】 図画面上に図形描画する場合の座標値計算では、以下の定義をしておくことを推奨する。

```
#define X_MAX 99
#define Y_MAX 63
#define X_MID (X_MAX+1)/2
#define Y_MID (Y_MAX+1)/2
```



ドットマトリックス・ディスプレイ（２）

主な描画・表示機能は、以下のようなものがあり、一覧表ならびにパラメータの詳細を記載しておく。

「ClearScreen()」は、ディスプレイ表示をクリア。
「NumOut(x,y,number)」は、座標値（ｘ，ｙ）に、数字（number）を表示
「TextOut(x,y,string,bool)」は、座標値（ｘ，ｙ）に、文字列（string）を表示、bool=true（画面消去あり）。
「GraphicOut(x,y,filename)」は、座標値（ｘ，ｙ）に 、ビットマップ(「.ric」)ファイルを表示。
「CircleOut(x,y,radius)」は、座標値（ｘ，ｙ）を中心とした半径（radius）で円を描きます。
「LineOut(x1,y1,x2,y2)」は、座標点(ｘ１，ｙ１)から（ｘ２，ｙ２）までの線分を描きます。
「PointOut(x,y)」は、座標点（ｘ，ｙ）にドットを描画します。
「RectOut(x,y,width,height)」は、座標点（ｘ、ｙ）から右幅（width）と高さ（height）の長方形を描きます。
「ResetScreen()」は、スクリーンをリセットします。

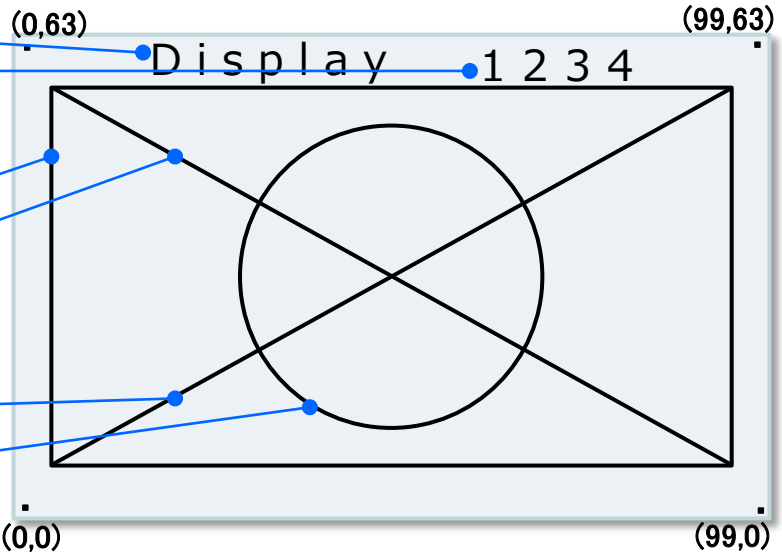
コマンド	引　数	備　考
ClearScreen		画面クリア
NumOut	X座標、Y座標、数字	整数値を出力
TextOut	X座標、Y座標、文字列、Bool	文字列を出力(Bool=true：画面消去)
GraphicOut	X座標、Y座標、ricファイル名	ビットマップ・ファイルを出力
CircleOut	X座標、Y座標、半径	円の作図
LineOut	X1座標、Y1座標、Ｘ２座標、Y2座標	線分の作図
PointOut	X座標、Y座標	点の作図
RectOut	X座標、Y座標、幅、高さ	長方形の作図
ResetScreen		スクリーンのリセット（画面クリア）

ドットマトリックス・ディスプレイ（３）

```
#define X_MAX 99    //  Program12.3
#define Y_MAX 63
#define X_MID (X_MAX+1)/2
#define Y_MID (Y_MAX+1)/2
task main()
{
  int  i = 1234;
  TextOut(15, LCD_LINE1, "Display", true);
  NumOut(60, LCD_LINE1, i);
  PointOut(1, Y_MAX-1);   // 左上点
  PointOut(X_MAX-1, Y_MAX-1);  //  右上点
  PointOut(1,1);              //  左下点
  PointOut(X_MAX-1, 1);  //   右下点
  Wait(200);
  RectOut(5, 5, 90, 50);   //  長方形
  Wait(200);
  LineOut(5, 5, 95, 55);   //   斜線
  Wait(200);
  LineOut(5, 55, 95, 5);   //   斜線
  Wait(200);
  CircleOut(X_MID, Y_MID-2,20);  //  円
  Wait(800);
  ClearScreen();    //  画面クリア
  GraphicOut(30, 10, “Smile 01.ric");
  Wait(500);
  ClearScreen();   //  画面クリア
  GraphicOut(30, 10, “Smile 02.ric");
  Wait(1000);
}
```

画面範囲、中心点設定

【参考】 LCD_LINE１～LCD_LINE8は、画面上からの行数位置のこと



【注意】 図形ファイル(.ric）は、NXT上に存在するファイルとなる。NXT上のファイル内容は、メニューバーの「tools」→「NXT Explorer」で確認できる。

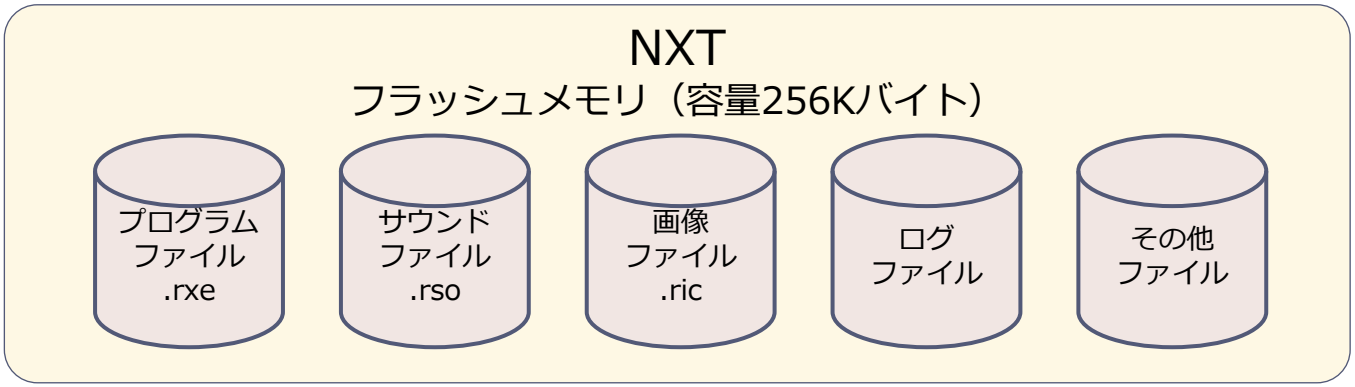
ファイルシステム（１）

NXT は、内臓された**フラッシュメモリ**に対して、ファイルの書き込みや読み込みができる。

このファイルの読み書きの機能を利用し、**センサーデータによるログ**をとったり、ロボットを動かす**プログラムを保存**したりすることができる。

ファイルの制限は、このフラッシュメモリの容量に依存する。

NXTの**API**（Application Programming Interface）**機能**では、**ファイル管理**（作成、名前変更、削除、検索）、それに**文字列**や、**数値**および**シングルバイト**でのファイルの読み込みと書き出しができる。



【注意】NXTフラッシュメモリーの容量は、256Kバイト。

ファイルシステム（２）

コンパイルしたプログラムをダウンロードする場合、すでにNXT側に同じファイル名があれば削除して、ダウンロードを行う。（本来であれば、既存ファイルの存在をチェックし、マニュアル操作で削除するか、自動的に他の名前に変更することで対応する）

ただこの場合は単純に上書きするが、特に問題は起きることはない。

ファイルを作成するには、「**Createfile(ファイル名, サイズ, ハンドル)**」を使って、ファイルを作成する。この場合の引数のサイズは作成するファイル容量でバイト数となる。またハンドルは、ファイルの識別IDとなり、NXT自体が独自に生成する。（NXTファームウェア内で独自に管理する数字）

ファイルへの書き込みは、「**WriteLnString(ハンドル, 文字列,　カウント)**」などを使って、ファイルのハンドルと文字列を入力する。ここでの引数のカウントは文字列のカウント数で出力となる。

その他、ファイルのクローズは、「**CloseFile(ハンドル)**」を利用する。また、ファイル名変更では、「**RenameFile（現ファイル名、新ファイル名）**」を利用する。

【注意】つぎのファイル操作(ファイルへの書き込み、読み込み)をするまでファイルはクローズしておく必要がある。再度読み込むときに「**OpenFileRead(ファイル名、サイズ、ハンドル)**」を使ってオープンする。また、ファイルを削除するか、名前変更する場合には、ファイルをクローズしておく必要がある。

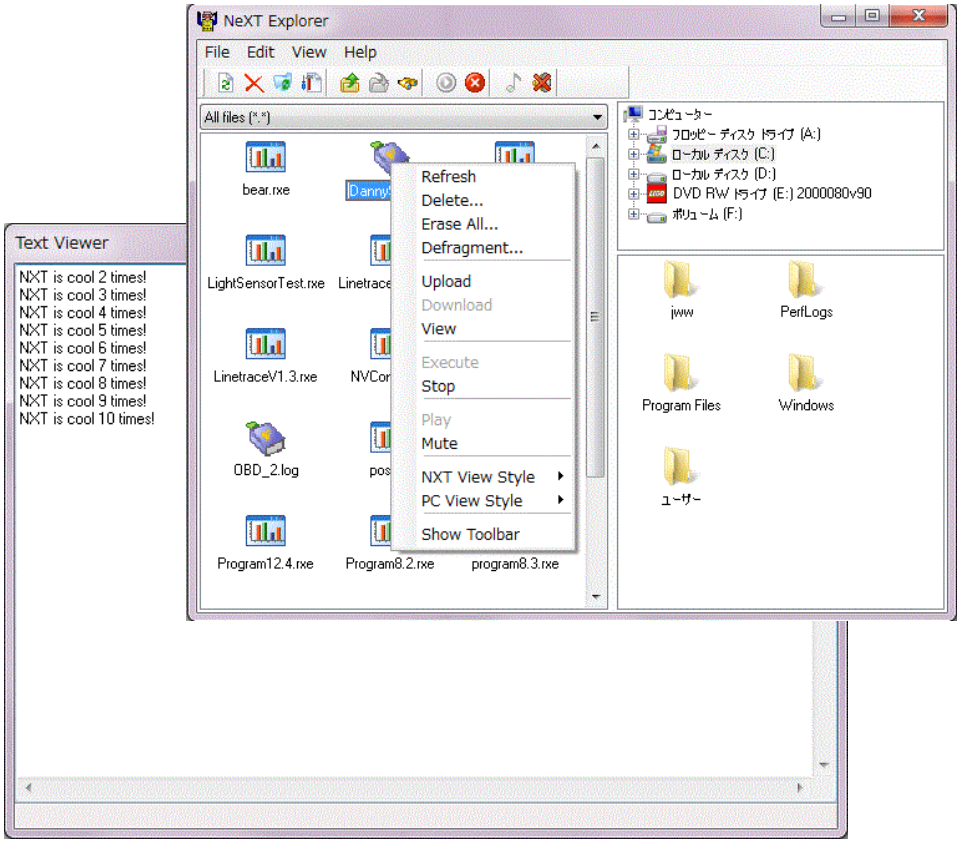
ファイルシステム（３）

つぎの例では、ファイルをどう作成し、文字列を書き込み、名前を変更するところを紹介してる。

```
#define OK LDR_SUCCESS  //  Program12.4
task main(){
  byte fileHandle;
  short fileSize;
  short bytesWritten;
  string read;
  string write;
  DeleteFile("Danny.txt");
  DeleteFile("DannySays.txt");
  CreateFile("Danny.txt", 512, fileHandle);
  for(int i=2; i<=10; i++ )
  {
    write = "NXT is cool ";
    string tmp = NumToStr(i);
    write = StrCat(write,tmp, " times!");
    WriteLnString(fileHandle, write, bytesWritten);
  }
  CloseFile(fileHandle);
  RenameFile("Danny.txt", "DannySays.txt");
}
```

結果はどうなっているだろうか。

それでは「**BricxCC**」から「**tools**」を選択し、さらに「**NXT Explorer**」へ行き、ファイル名「**Dannaysay.txt**」を、PCへアップロード（upload）して、中を覗いて見てみよう。



ファイルシステム（４）

```
//   Program 12.4
task main(){
   byte handle;
   if(CreateFile(“ASCII.txt”, 2048, handle) ==  NO_ERR)
  {
     for (int i=0; i < 256; i++) {
      string s = NumToStr(i);
      int slen = StrLen(s);
      WriteBytes(handle, s, slen);
      WriteLn(handle, i);
    }
    CloseFile(handle);
  }
}
```

これも単純なプログラムで、ファイルを作成し、エラーが無ければ、つぎに「NumToStr(i)」関数を使って、０から255までのコードによる文字変換を行い、「WriteBytes(handle,s,slen)」関数でこの文字に改行（キャリッジリターン）を追加してファイルに書き込んでいます。

この結果は、「ASCII.TXT」をオープンする際、テキスト・エディタ（例えばWindowsのNotepad）を使って見ます。この数値で書き込まれた内容は、16進数で書かれた数値も、アスキー（ASCII）コードに変換し、人にわかり易い文字列で見ることができます。

ここで２つの重要な関数で、ファイルから文字列を読み込む「ReadLnString」関数と、数字を読み込む「ReadLn」関数を紹介します。

ファイルシステム（５）

ここでの事例での最初の行では、「main」タスクで、乱数から文字列にしてファイルを作成する「**CreateRandomFile**」サブルーチンを呼び出しています。またこの行をコメントにしたり、手動作成によるテキストファイルを使ったりすることができます。

つぎに「main」タスクでこのファイルを読み込むためにオープンし、ファイルのエンドになるまで「**ReadLnString**」関数を使って読み続け、テキストを表示します。

「**CreateRandomFile**」サブルーチンは、乱数で発生させ、あらかじめ文字に変換して、それをファイルに書き込みます。

「**ReadLnString**」関数は、ファイル・ハンドルと文字列の変数を引数として、呼び出されたときに文字列をテキストラインに書き込み、リターン値としてエラーコードを返します。ファイルの読み込みが最終行になった場合には、このエラーコードで分るようになっている。

モジュール	引　数	説明
CloseFile	handle	ファイルクローズ
CreateFile	fname,fsize,**handle**	ファイル新規作成
OpenFile	Handle	ファイルオープン
OpenFileRead	fname,fsize,**handle**	ファイル読込オープン
DeleteFile	filename	ファイル削除
Read	handle,value	読込み
ReadBytes	handle,length,buf[]	バイト単位読込み
ReadLn	handle,value	行単位読込み
ReadLnString	handle,output	文字列単位読込み
remove	fname	ファイル削除
RenameFile	Oldfname,Newfname	ファイル名変更
rename	old,new	ファイル名変更
Write	handle,valu	書込み
WriteBytes	handle,vuf[], **cnt**	バイト単位書込み
WriteLn	handle,value	行単位書込み
WriteLnString	handle,str, **cnt**	文字列単位書込み
WriteString	handle,str, **cnt**	文字列書込み

ファイルシステム（６）

```
// Program12.6
#define FILE_LINES 10

sub CreateRandomFile(string fname, int lines){
  byte handle;
  string s;
  int bytesWritten;
  DeleteFile(fname);
  int fsize = lines*5; //create file with random data
  if(CreateFile(fname, fsize, handle) == NO_ERR) {
    int n;
    repeat(FILE_LINES) {
      int n = Random(0xFF);
      s = NumToStr(n);
      WriteLnString(handle,s,bytesWritten);
    }
    CloseFile(handle);
  }
}
```

ここでの事例での最初の行では、「main」タスクで、乱数から文字列にしてファイルを作成する「**CreateRandomFile**」サブルーチンを呼び出しています。またこの行をコメントにしたり、手動作成によるテキストファイルを使ったりすることができます。

つぎに「main」タスクでこのファイルを読み込むためにオープンし、ファイルのエンドになるまで「**ReadLnString**」関数を使って読み続け、テキストを表示します。

```
task main(){
  byte handle;
  int fsize;
  string buf;
  bool eof = false;
  CreateRandomFile("rand.txt", FILE_LINES);
  if(OpenFileRead(“rand.txt”, fsize, handle) == NO_ERR){
    TextOut(10, LCD_LINE2, "Filesize:");
    NumOut(65, LCD_LINE2, fsize);
    Wait(600);
    until(eof == true){ // read the text file till the end
      if(ReadLnString(handle,buf) != NO_ERR) eof = true;
      ClearScreen();
      TextOut(20, LCD_LINE3, buf);
      Wait(500);
    }
  }
  CloseFile(handle);
}
```

ファイルシステム（７）

最後のプログラムでは、ファイルから数値を読み込む方法を学ぶ。

ここでは、状況が複雑な場合の事例をご紹介しよう。最初のコードで、INT（整数）が設定されているが、マクロなどと混同しないように注意しよう。 これらは、プリプロセッサ・ステートメントと呼ばれる。

```
#ifdef  INT
     ....Code....
#endif
```

これによって、あらかじめ宣言した「**#define　INT**」で、以下の２つのステートメント（「**#ifdef**」と「**#endif**」）で挟まれたコードがコンパイルされる。もし、「**#define　LONG**」で宣言した場合には、「INT」のコードではなく、下の「main」タスクのある「LONG」のコードがコンパイルされることになる。

この手法は、同じ関数「**ReadLn（handle、val）**」を使って、ファイル読み込み時に、「int」（16ビット）と「long」（32ビット）の両方のうち一つのプログラムをコンパイルすることになる。

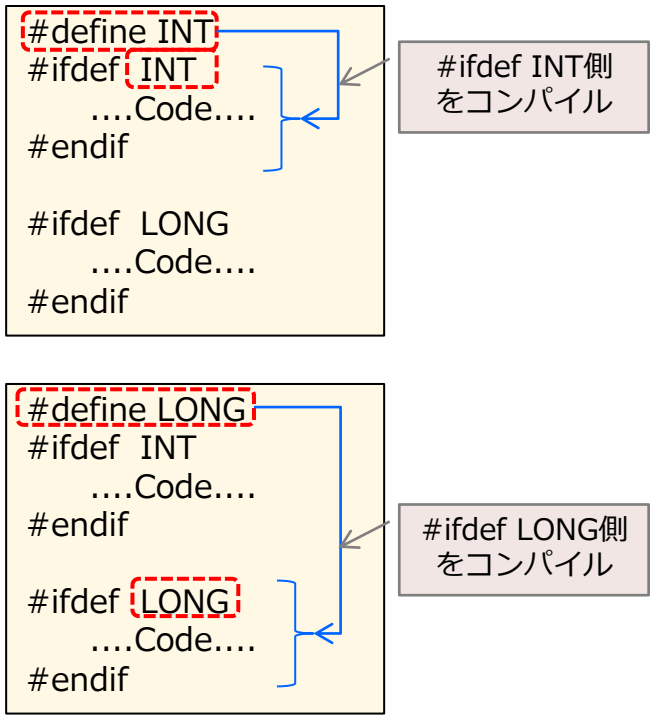
ここでの２つの引数は、前述したファイル・ハンドルと数値変数で、リターン値はエラーコードとなる。

この関数では、「int」で宣言された場合には、2バイトごとにファイルから読み込むことになる。

また「long」で宣言された場合には、4バイトごとにファイルから読み込むことになる。

さらに「bool」（ブーリアン）値を使って同様に、読み込み、書き出しもできる。

#ifdef プリプロセッサ



ファイルシステム（８）

#define INT
で#ifef INT側をコンパイル

```
// Program12_7
#define INT // INT 又はLONG
#ifdef INT
task main () {
  byte handle, time = 0;
  int n, fsize, len, i, in;
  DeleteFile("int.txt");
  CreateFile("int.txt", 4096, handle);
  for(int i = 1000; i<=10000; i+=1000)
      {WriteLn(handle, i); }
  CloseFile(handle);
  OpenFileRead("int.txt", fsize, handle);
  until(ReadLn(handle, in) != NO_ERR){
    ClearScreen();
    NumOut(30, LCD_LINE5, in);
    Wait(500);
  }
  CloseFile(handle);
}
#endif
```

ここでのプリプロセッサを利用するメリットは、コンパイルの場合分けができ、プログラムのコンパイルした実行コードを小さくすることにある。

よって、実行コード名は同じになるが、どちらでコンパイルされたかは、利用時点で利用者が分かっておく必要がある。

```
#ifdef LONG
task main () {
  byte handle, time = 0;
  int n, fsize,len, i;
  long in;
  DeleteFile("long.txt");
  CreateFile("long.txt", 4096, handle);
  for(long i = 100000; i<=1000000; i+=50000)
      { WriteLn(handle, i);}
  CloseFile(handle);
  OpenFileRead("long.txt", fsize, handle);
  until(ReadLn(handle, in) != NO_ERR){
    ClearScreen();
    NumOut(30, LCD_LINE5, in);
    Wait(500);
  }
CloseFile(handle);
}
#endif
```

ファイルシステム（９）

補足として、複数ロボットでのリモートによるファイル読み書きなどのモジュール群も紹介しておく。つまり、マスターのNXTから、スレーブのファイルの読込みや書込み、さらにファイルのオープン、クローズ、削除、名前変更などができるモジュール群が豊富に用意されている

モジュール	引　数	説明
RemoteCloseFile	conn, handle	スロットconnのファイルクローズ
RemoteDelateFile	conn, filename	スロットconnのファイル削除
RemoteOpenRead	conn,filename,handle,size	スロットconnのファイル読込みオープン
RemoteOpenWrite	conn,filename,size,handle	スロットconnのファイル書込みオープン
RemoteMessageRead	conn,queue	スロットconnのメッセージ読込み
RemoteMessageWrite	conn,queue,msg	スロットconnのメッセージ書込み
RemoteRead	conn,handle,numbytes,data[]	スロットconnの読込み
RemoteWrite	conn,handle,numbytes,data[]	スロットconnの書込み
RemoteRenameFile	conn,oldname,newname	スロットconnのファイル名変更
RemoteSetBrickName	conn,name	スロットconnのNXT名称変更

【展開】

ここでは、さまざまなNXC APIモジュールを紹介した。

これまで紹介してきたAPIにどのようなものがあるか、一覧表をまとめておく。

1. タスクおよびタイミング関数
2. 文字列関数
3. 配列関数
4. 数式関数
5. 入力関数
6. 出力関数
7. サウンド関数
8. IO制御モジュール関数
9. LCD画面表示関数
10. ファイルI/O関数
11. ボタン関数
12. ユーザインタフェース関数
13. Lowspeed I2C関数
14. Bluetooth関数
15. USB関数

まとめ

- この章ではNXTが提供する拡張機能である、高性能のタイマー、NXT本体の画面表示、およびファイルシステムについて学んだ。
- 高機能タイマーを使って、正確な時間制御が可能となる。
- NXT本体の画面表示は、プログラムの動きを画面表示で見ることができ、プログラムディバッグなどでも便利に利用できる。
- ファイル・システムは、センサーデータの取得とファイル出力が可能となったり、サウンドファイルの読込みによる再生なども可能となる。
- より複雑なプログラムができることを学んだ。

Ⅳ.　カリキュラム編

BricxCCおよびNXCを使った
NXTロボット教育を行うために

１．ロボット教育カリキュラム

教育カリキュラムは、一定の教育の目的に合わせて、考え出された教育内容とその決まった修業年限の間での教育と学習を総合的に計画すること（Wikipediaより）。

ロボット教育による基本的は学習は、

１）機械系の組み立て、機構関連、それにモータ動力などが学びとれるもの。

２）電子・電気によるLEDやセンサ技術が学びとれるもの。

３）情報技術による組込み・制御のプログラミングが学びとれるもの。

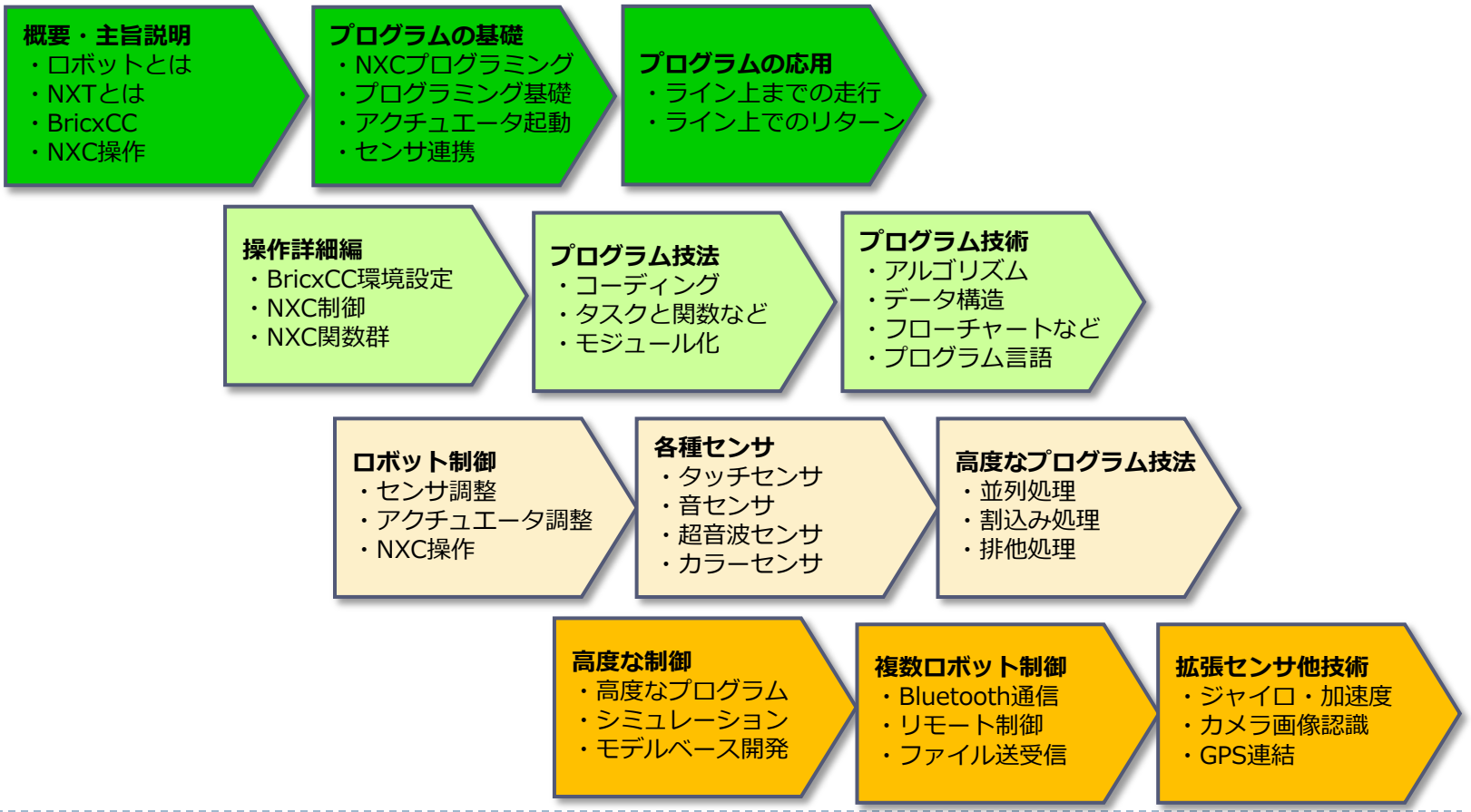
などがある。

ここでは、３）についてのロボット教育を中心としてまとめている。

① ロボット教育カリキュラムとは

教育カリキュラムは、一定の教育の目的に合わせて、考え出された教育内容とその決まった修業年限の間での教育と学習を総合的に計画（Wikipediaより）。

さまざまな教育コンテンツから、何を学びとらせるか？　→ 楽しく学べるものは何か？　学習者のスキルなどに合わせた現場での判断が重要



② Mindstorms学習カリキュラム研究例

「ロボットを使った論理的思考力を育むカリキュラムの開発（川原田康文）」から
LEGO Mindstorms のSTEP学習の基礎基本の各課題と指導の視点

STEP	課題内容	指導の観点
1	基本ロボットの作成	構造の理解
2	４秒前進して停止する	ロボットの動きとモータ回転方向
3	６秒間前進して、２秒間後退し、止まる	［STEP２の発展］
4	円を描く	左右のモータの出力の違いによる動き
5	3秒間前進し、右に９０度曲がり、止まる	９０度曲がるしくみ
6	正方形を描く	［STEP５の発展］前進＋９０度回転＋前進＋・・・・
7	STEP６のプログラム改良し何度も繰り返す	［STEP６の発展］繰り返しの概念
8	前進し、黒い線にぶつかったら止まる	光センサの概念
9	線の右側をたどりながら動くプログラム	ラインの右側をたどるための概念とプログラム
１０	線の両側をたどりながら動くプログラム	［STEP９の発展］

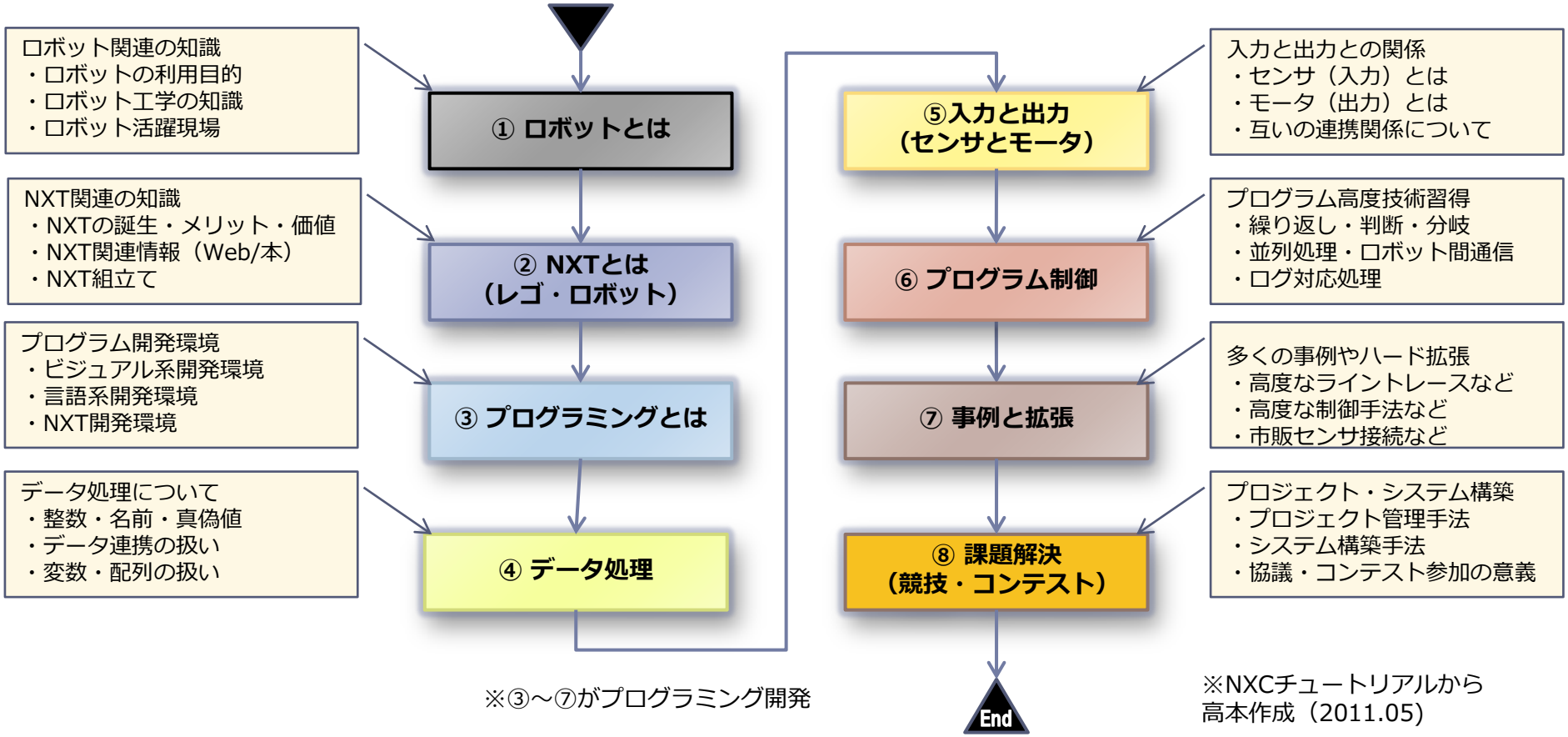
何を学習させるか？

これだけでも、何度も繰り返すプログラムの改良で、
組込み・制御を学ぶことができる。

2．ロボット教育カリキュラム思考

ここでは、「ロボット教育カリキュラム」のひとつの提案を行っている。
以下のフローは、情報技術関連知識をベースとした教育コンテンツとその教育課程を分類して、提示している。

実際の教育現場では、時間割に応じて、取捨選択して組合わせることができると考えている。
是非とも、生徒が興味が湧き、楽しく学べる内容と場を提供してみては如何だろうか？



※③～⑦がプログラミング開発

※NXCチュートリアルから
高本作成（2011.05）

①ロボットとは

ロボットを学ぶ上での広範囲な知識は幅広く、何を知るべきか、何を教えるべきかを明確にした上で、学んでもらうことが必要では・・・・

ロボット関連の知識とは何かを教える。
特に
① ロボットの利用目的
② ロボット工学の知識
③ ロボット活躍現場
など

高本作成（2011.05）

項目	カテゴリ	教育内容
1	ロボットの基礎知識	・ロボットとは何か？・ロボットの目的 ・ロボットの種類、・ロボットと機械の違い ・ロボットのメリット ・ロボットの形状（人間型・非人間型）
2	ロボット工学	・基礎数学、物理学、運動学、力学ほか
3	ロボット技術	・コンピュータ・プログラム ・アクチュエータ・センサの制御・組込み
4	ロボットの実用現場	・家庭内ロボット、医療・介護ロボット ・産業用ロボット、調理ロボット、ロボットカー ・ホビーロボット、癒しロボット、・軍事用ロボット ・エンターテイメント用ロボット
5	ロボット文化・歴史	・小説上のロボット、映画上のロボット ・アニメ・漫画上のロボット ・エンターテイメント

参考
1）新星出版社「ロボットのしくみ」(2009.04)
2）オーム社「絵ときでわかるロボット工学」川嶋健嗣著（2007.05)

② NXTとは（レゴ・ロボット）

なぜロボットを学ぶ上で、LEGO Mindstorms NXTが選択されるのか？
① 教育用として MIT（米国マサチューセッツ工学）にて開発され、教育現場での利用実績も多いこと
② 学習内容も充実していて、参考事例もWebや出版を通じて豊富であること
③ 多くの競技会やコンテストなどが開催され、教育の場が広がること
④ 学習する知識や知恵も豊富に対応でき、カリキュラムも簡易なものから高度なものまで対応可能なこと

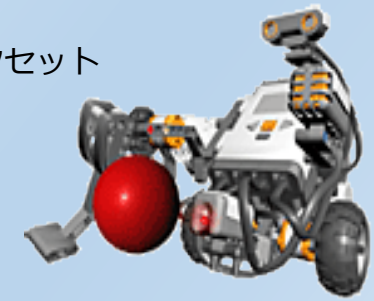
■NXTのメリット・価値
・子供から大人までの学習キット
・機構からコンピュータ制御まで　通じてロボット工学の教材
・豊富な教材（参考本・Webサイト）
・簡易なものから高度・複雑なものまで幅広く対応
・基本的な学習を短時間で学べる
・玩具として楽しく学べる　などなど

■NXTの歴史
★1998年に LEGO Mindstorms RCX 販売開始
＞開発環境は、ROBOLABが付属（別売）
★2006年に LEGO Mindstorms NXT 販売開始
＞その他教育版（日本語化対応）が販売される
＞開発環境は、NXT-SW（ビジュアル系）が付属（別売）
★2009年に LEGO Mindstorms NXT2.0 販売開始
＞教育版と玩具版が販売される。
＞玩具版は、英語版のみ

■NXT関連情報（Web/本）
参考【添付資料】

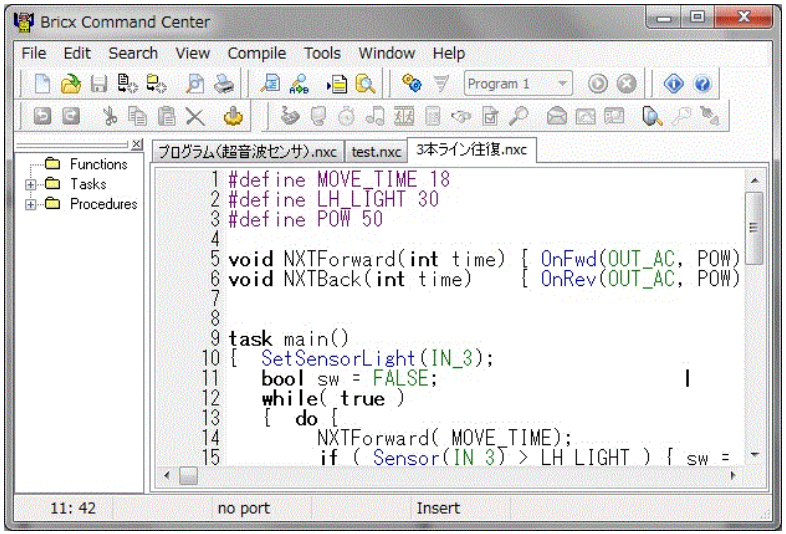
■NXTとPCとの接続について
・ USB接続/Bluetooth接続

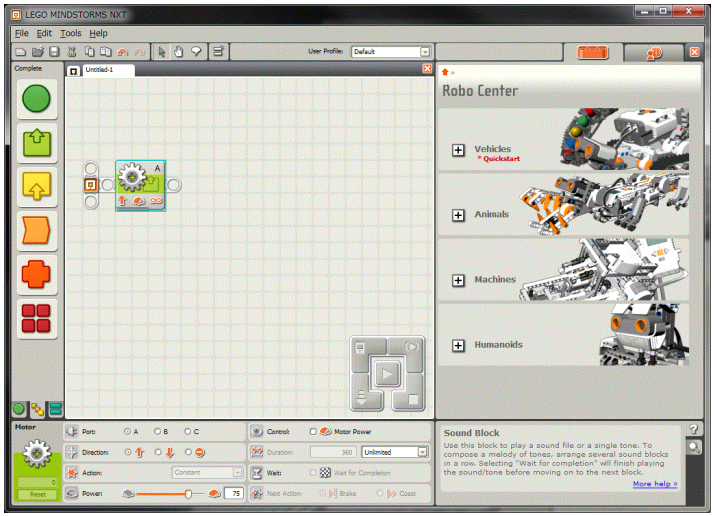
■NXT組立て
簡単な組み立てブロックセット
分かり易い作成手順図
豊富なサンプル



③ プログラミングとは

プログラミングにおいては、その環境設定から、基本的な使い方（文法など）の理解が必要となる。
情報処理技術の基本的なものを学ぶ必要がでてくる。
＜左図は統合開発環境のBricxCC画面、右図がNXT-G画面＞





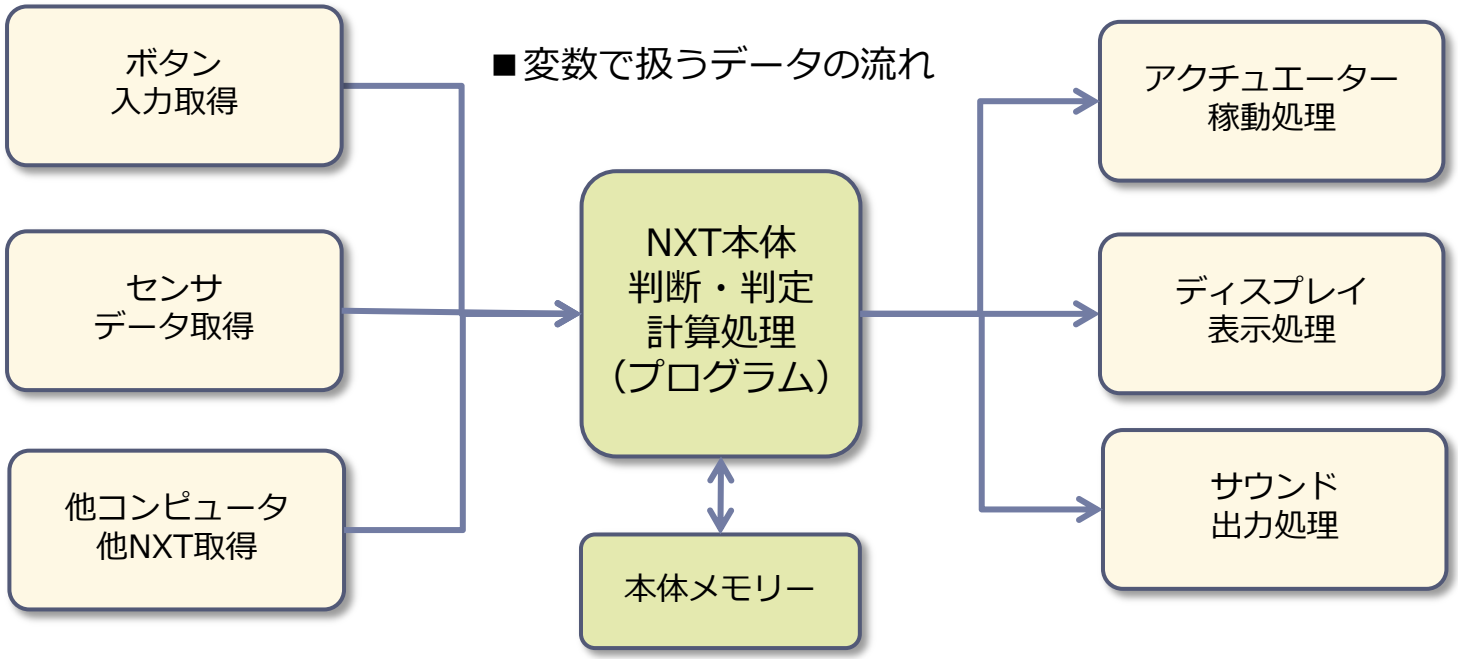
備考
※ LEGO MINDSTORMS NXTの開発環境は豊富に提供されていて、オープンソースなど、Web上でも多くの事例サンプルが紹介されている。

④ データ処理

プログラミングでは、データの処理の理解も不可欠となる。NXTで扱うデータは、整数、論理、それと文字列の３つのタイプ（型）を持つ。センサから取れるデータや、モータを動かすときに使うデータ、さらにはセンサ間で利用する値を連携して利用するなど、理解する必要がある。

■レゴで扱えるデータは、３種類
1） 整数（int, longなど）
2） 文字列は、テキスト文字列
3） 論理は、真偽（True=1,False=0）
<別途、配列も扱える>

■変数・定数の扱い
1） グローバル宣言
2） グローバル定数
3） グローバル変数、ローカル変数など



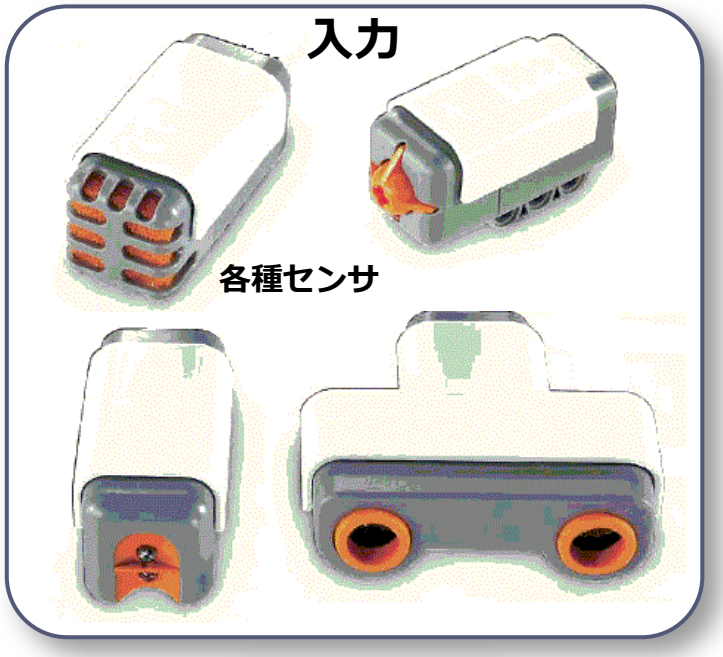
⑤入力と出力（センサとモータ）

■入力と出力との関係
・センサ（入力）とは
・モータ（出力）とは
・互いの連携関係について

■様々な入力と出力のプログラミング

タッチセンサが障害物に接触

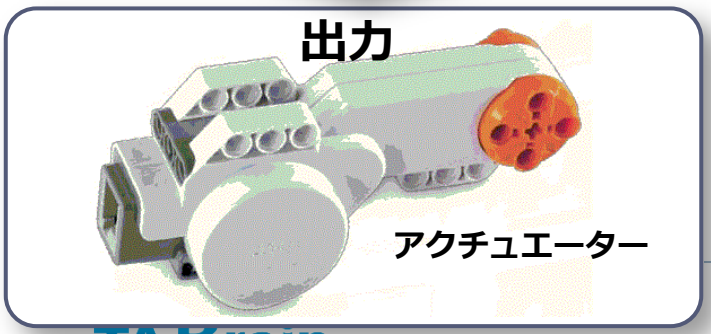

```
task main()
{
  SetSensorTouch(IN_1);
  OnFwd(OUT_AC, 75); ①
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      OnRev(OUT_AC, 75);②
      Wait(300);
      OnFwd(OUT_A, 75);③
      Wait(300);
      OnFwd(OUT_AC, 75);④
    }
  }
}
```



システムとは
①入力
＋
②処理
＋
③出力
で構成

処理

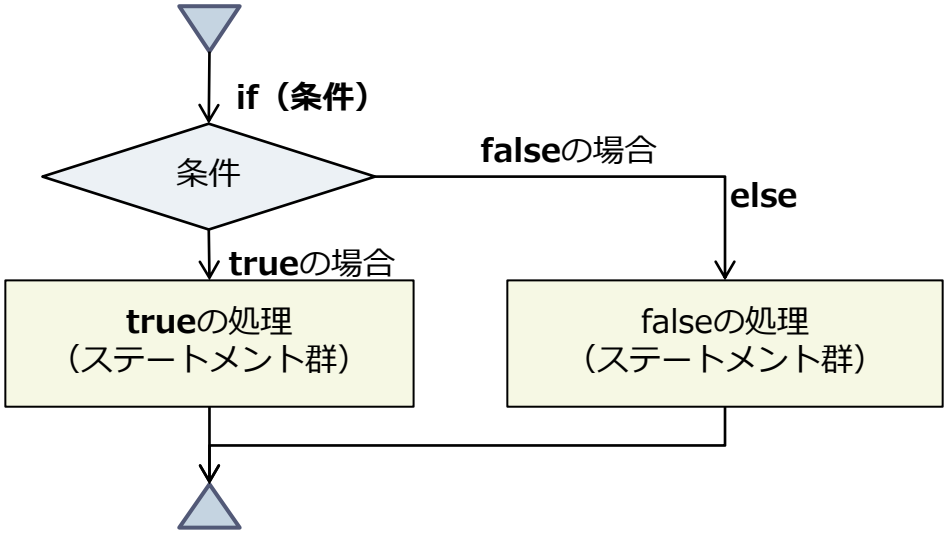
主に制御構造（if,do,while,repeatなど）を使ってコンピュータ制御を実施



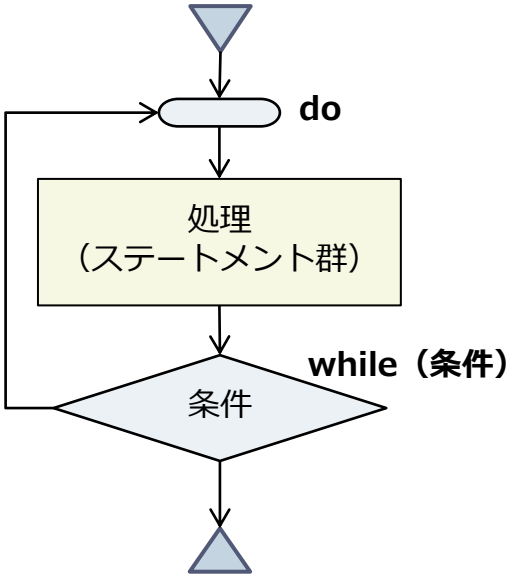
※アクチュエーターとは、モータを回すといった制御を行う機械

⑥ プログラム制御

プログラミング学習の上で、初心者から中級者になる上では、さまざまなプログラムの文法や、特殊な処理内容を理解していく必要がある。
ここでは、モデル化として、最もベーシック（基本的）なフローチャートを示している。
現場の教育レベルに応じて内容を編集対応していく必要ある。
プログラム処理での**制御構造**を使った**アルゴリズム**処理が重要となる。
※制御構造とは、if、do、repeat、whileなどとなる。



if文のフローチャート表現



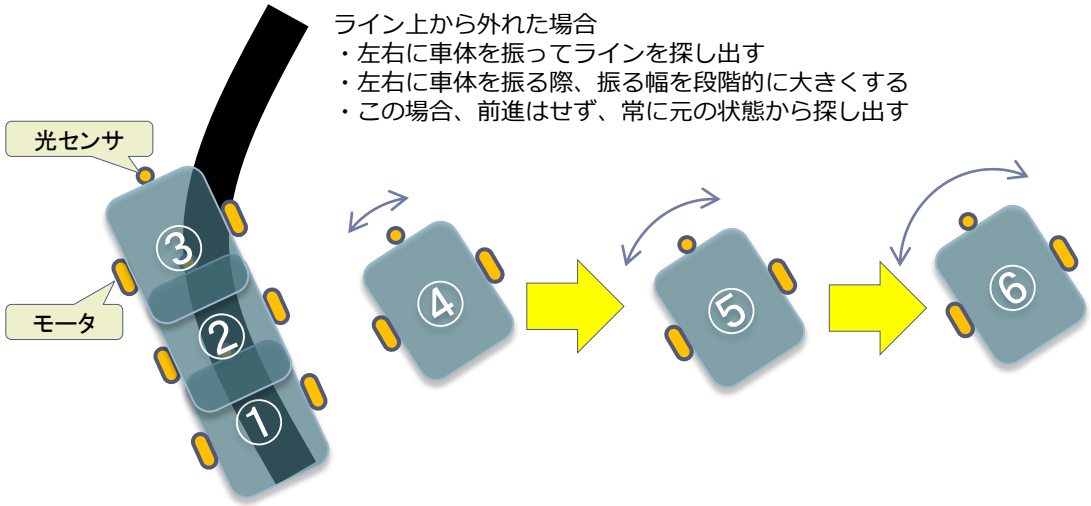
do文のフローチャート表現

多くの事例やハード拡張
・高度なライントレースなど
・高度な制御手法など
・市販センサ接続など

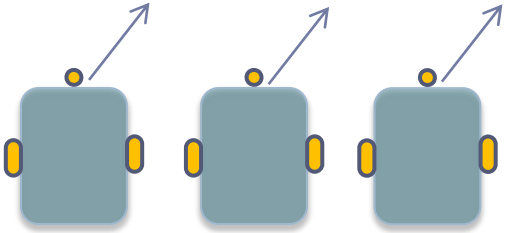
⑦ 事例と展開・拡張

NXTをさらに上級レベルまで引き上げるためのもので、より複雑で、高度なテクニックを学ぶことが可能。
必要に応じて、ハイレベルな教育展開も用意可能。

■ライントレースプログラム

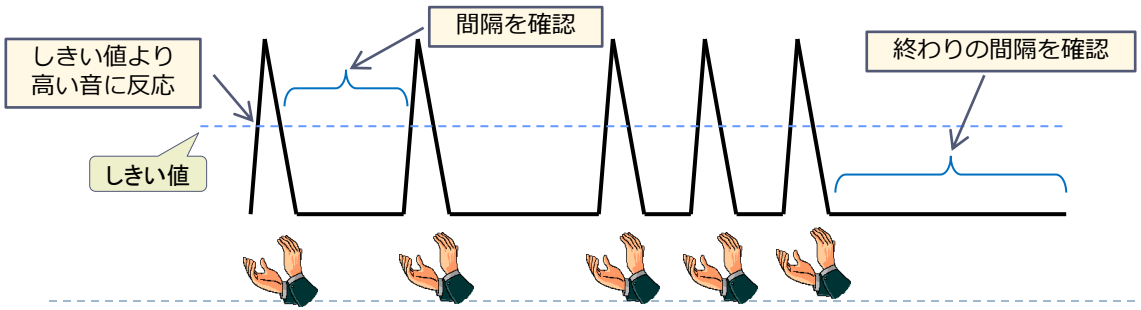


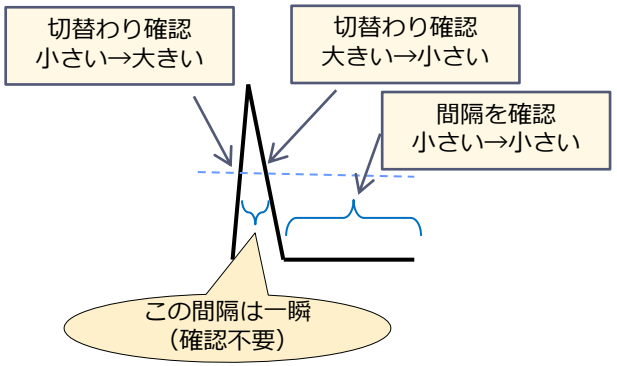
■群ロボット制御



複数のロボット群が、同期を取りながら
（無線を通じて）動きを制御する。
※無線操作技術などが重要

■音センサによるのトレース（手拍子ものまね）





⑧ 課題解決（競技・コンテスト）

競技会・コンテストによって、子供達は多くのことを学ぶことが可能

- **コミュニケーション能力・協調性**
- **プレゼンテーション能力・企画力・提案力**
- **プロジェクト推進・プロジェクト管理・PDCA**
- **モノづくりの楽しさ・喜び**
- **独創性や創造性**

■LEGO社公認の競技会

・ロボカップジュニア
　ダンス、サッカー、レスキュー

FLL FIRST LEGO League

WRO World Robot Olympiad

■ETロボコン

日本の産業競争力に欠くことのできない重要な「組込みシステム」分野における技術教育をテーマに、決められた走行体で指定コースを自律走行する競技です。同一のハードウェア(LEGO MindstormsTM)に、 UML等で分析・設計したソフトウェアを搭載し競うコンテストです。※UML（Unified Modeling Language）

■その他競技会

　その他多くの競技会が開催されているサイトを検索できる

あとがき

- 2011年3月11日の大震災による福島原発で、海外製のロボットが多く導入された。　何故日本製は利用されなかったのか？
- 何故、東電は原発の安全性を過信して、ロボット導入をしなかったのか？
- 日本で使われない日本製ロボットが他の国に使われるのは何故か？
- 何故の日本のロボット研究では人間型が多いのか？
- 何故ロボット産業が爆発的に普及して行かないのか？
- 何故日本のロボット研究者が海外へ活路を見出して出ていくのか？

- 今後、「ロボット活用見直し」で重要視されるのでは・・・・
- さらに、もっと補助金・交付金がロボット関連で予算が付くのでは？

そして、日本のロボット技術力で、再度モノづくり日本を再生させられるのでは・・・・
いや、日本再生を即進めることが重要だ・・・・・

【参考資料】

① 参考資料１：カリキュラム導入・運用シナリオ

１．体験

・体験セミナーなどへの参加：具体的な教育内容を把握

２．試用

・評価版やチュートリアル版などを導入し、試験的に利用（課題など抽出）

３．選定

・類似のもの含め評価し、最も現場にあったものを選定

４．導入

・教材の提供を受けたり、導入事例を参考にしていく

５．習得

・教育トレーニングの収得や、講習会への参加などを実施

６．カリキュラム検討

・教育収支・目的、内容のまとめ。教材の準備、教師用テキスト準備

７．カリキュラム実行

・実行段階での新築管理、教育レベルの評価、フィードバック。競技導入

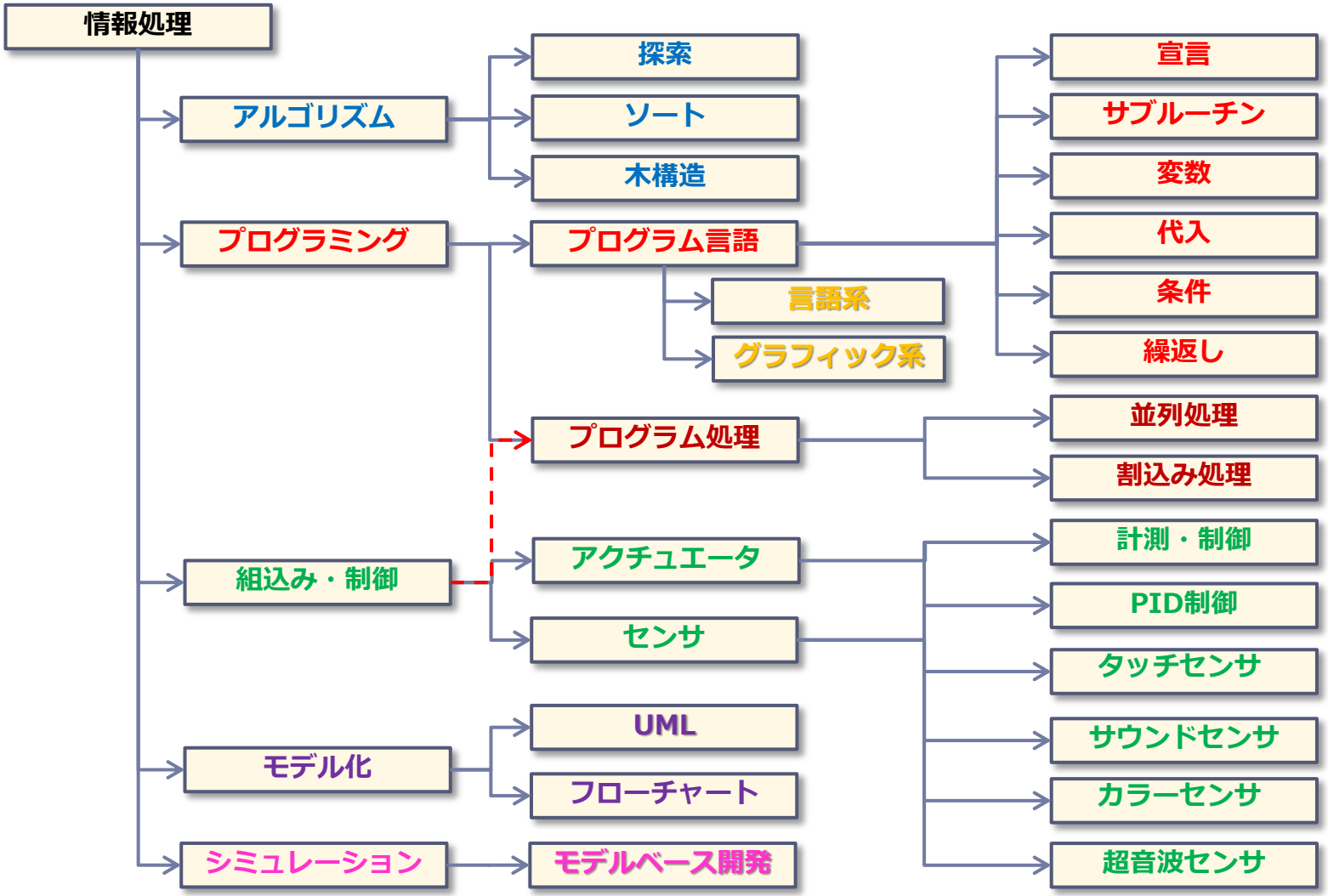
８．コンテスト・協議会

・実行段階での新築管理、教育レベルの評価、フィードバック。競技導入

９．認定試験習得

・プログラム開発における認証試験協会が存在

② 参考資料２： ロボット教育で学ぶ情報技術



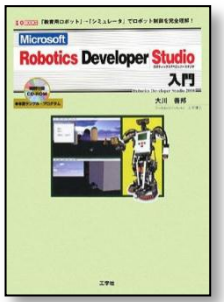
レゴ・マインドストームでもこれだけの教育ができます。

③ 参考資料３： LegoMindstorms NXT関連本

初心者向け紹介本

開発者向け紹介本

ロボット開発者向け紹介本

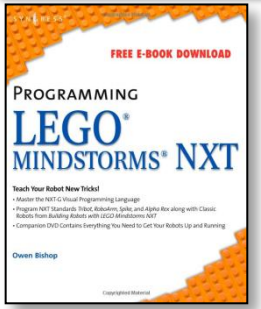


ロボット開発者向け紹介本

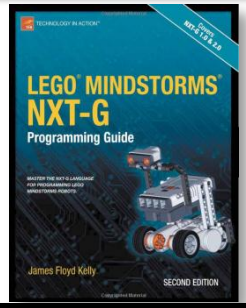
ロボット開発者向け紹介本

ロボット開発者向け紹介本

機構設計者向け紹介本

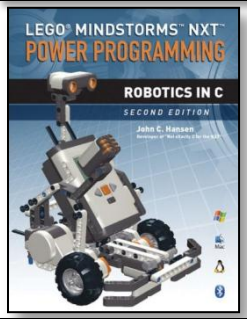


プログラム開発向け紹介本

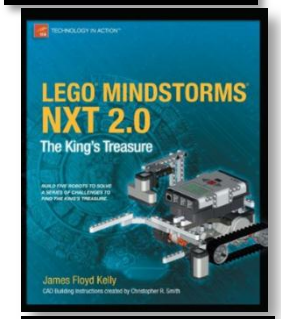


プログラム開発向け紹介本

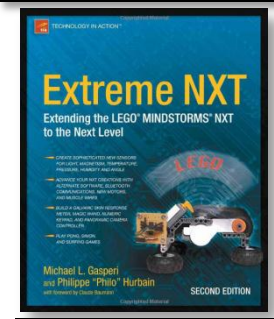
学生向け紹介本



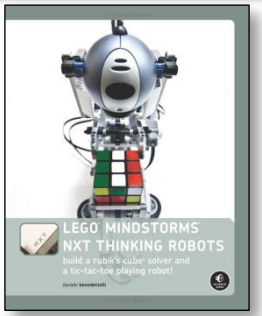
組込ソフト開発者向け紹介本



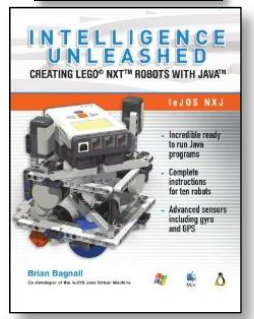
機構設計者向け紹介本



NXT拡張者向け紹介本



マニュアック向け紹介本



JAVAによる開発紹介本

著者が保有する参考本

④ 参考資料４： BricxCCとNXCで学ぶ 知的LEGO Mindstorms NXTプログラミング入門

2012年5月1日発刊　CQ出版社　￥2,600（税別）
大学・高専・高校向けプログラミング入門書

I. 導入・準備編

第1章 ロボット教育
1.1　ロボット教育とは
　(1)実社会でのロボットの活躍
　(2)ロボットの研究
　(3)ロボット教育が目指すもの
　(4)学校でのロボット教育
1-2　レゴ・マインドストームによるロボット教育
　(1)NXTハードウェア仕様
　(2)NXTプログラミング教育
　(3)レゴ・マインドストームの教育

第2章　NXTの購入と開発環境
2-1　製品と購入について
2-2　NXTを動かす手順
2-3　ロボットの組み立て
2-4　NXTのソフトウェア開発環境
2-5　BricxCC・NXCとは

第3章　BricxCC統合開発環境
3-1　BricxCCのプログラミング開発環境構築
　(1)BricxCCのインストール
　(2)BricxCCの起動
　(3)BricxCCの画面概要
3-2　BricxCCの基本的な環境設定
　(1)環境設定「General」(一般)
　(2)環境設定「Editor」(エディタ関連)
　(3)環境設定「Compiler」(コンパイラ関連)
　(4)環境設定「API」(アプリケーション開発インターフェース)
　(5)環境設定「Start Up」(スタートアップ)
　(6)環境設定「Templates」(テンプレート関連)
　(7)環境設定「Macros」(マクロ関連)
　(8)環境設定「Color」(カラー設定)
　(9)環境設定「Options」(オプション)
3-3　BricxCCの主な機能
　(1)File(ファイル)機能
　(2)Edit(編集)機能
　(3)Search(検索)機能
　(4)View(ビュー)機能
　(5)Compile(コンパイル)機能
　(6)Tools(ツール)機能
　(7)Window (ウィンドウ)機能
　(8)Help(ヘルプ)機能

II. 基礎編

第4章　ロボットを動かす　初心者向き
4-1　ロボットの前進と後退
　(1)BricxCCを使ってNXCプログラムの作成
　(2)NXCプログラムの保存とコンパイル
　(3)NXTロボットへのプログラムのダウンロード
　(4)NXTロボットのプログラム実行
4-2　ロボットの停止
4-3　スピードの変更
　【課題1】定位置で停止
　【課題2】正確に前進・後退
　Column4-1　プログラムのデバッグ　情報技術習得

第5章　ロボットを複雑に動かす　初心者向き
5-1　ロボットの旋回(簡単なプログラミング)
5-2　繰り返しの動き(わかりやすいプログラミング)
5-3　渦巻いて動く(複雑なプログラミング)
5-4　ランダムに動く(高度なプログラミング)
5-5　センサを使う(知的なプログラミング)
5-6　ディスプレイとサウンドを使う(賢いプログラミング)
　【課題3】直角に曲がる
　【課題4】円を描く
　Column5-1　わかりやすいプログラム　情報技術習得

結構重要

第6章　NXCプログラム仕様
6-1　記述ルール
　(1)空白文字の扱い
　(2)コメント記述
　(3)数値定数
　(4)文字列定数と文字定数
　(5)識別子とキーワード
　(6)演算子(オペレータ)
6-2　変数とデータ構造
　(1)データ型(タイプ)と変数宣言
　(2)構造体(struct)
　(3)配列(array)
　(4)定数変数(const)
　(5)静的変数(static)
　(6)列挙タイプ(enum)
　(7)タイプ宣言(typedef)
6-3　制御ステートメント群
　(1)repeatステートメント
　(2)while ステートメント
　(3)for ステートメント
　(4)ifステートメント
　(5)do-whileステートメント
　(6)untilステートメント
　(7)switchステートメント
6-4　コード群
　(1)タスク
　(2)関数とプロシジャ
　(3)プリプロセッサ
6-5　コンパイルと実行ファイル
　(1)ソース・ファイルと実行ファイル
　(2)コンパイル
　【課題5】車庫入れ
　【課題6】プログラムの変更
　Column6-1　プログラム設計とモデル化　情報技術習得

④ 参考資料４： BricxCCとNXCで学ぶ 知的LEGO Mindstorms NXTプログラミング入門

III. 応用編

中級者向き

第7章　センサを制御する　重要

7-1　センサで判断する
　(1)NXT付属のセンサ群の機能
　(2)センサの設定とセンサ値の利用
7-2　センサ設定とデータ値取得方法
　(1)センサ設定1：SetSensorによる入力ポートの設定方法
　(2)センサ設定2：タイプ設定とモード設定
　(3)センサ設定3：各種センサ設定関数
　(4)センサ値の取り出し方法
7-3　タッチ・センサを使いこなす
7-4　音センサを使いこなす
7-5　光センサを使いこなす
7-6　カラー・センサを使いこなす
7-7　超音波センサを使いこなす
　【課題7】三重線の往復
　【課題8】手拍子の記録と再生　情報技術習得
　Column7-1　データ構造とアルゴリズム

上級者向き

第8章　高度なモータ制御　重要

8-1　調整機能をもつモータ制御(OnFwdReg)
8-2　同期をとるモータ制御(OnFwdSync)
8-3　回転角度によるモータ制御(RotateMotorEx)
8-4　モータのPID制御(OnFwdRegPID)
8-5　回転センサを使いこなす(MotorRotationCount & MotorTachoCount)
　【課題9】ライン・トレース
　【課題10】正確に直角に曲がるロボット
　Column8-1　メモリ管理と高速化　情報技術習得

上級者向き

第9章　並列処理と割り込み　重要

9-1　マルチ・タスク(並列処理)
9-2　割り込み処理と衝突(コンフリクト)
9-3　セマフォによる制御
9-4　mutex変数による制御
9-5　safecallによる同期処理
　【課題11】二つのセンサによる割り込み処理
　【課題12】割り込み処理を考慮したライン・トレースの挑戦
　Column9-1　組み込みプログラミング　情報技術習得

第10章　便利機能

10-1　LCD画面表示機能
　(1)テキスト表示
　(2)グラフィック表示
　(3)画像ファイル表示
10-2　ファイル入出力機能
　(1)ファイルの入出力機能
　(2)センサ・データ値のファイル入出力事例
10-3　サウンド機能
　(1)メロディを奏でる
　(2)サウンド・ファイルの変換と再生その他
10-4　ロボット間通信機能(複数ロボットの操作)　上級者向き
　(1)NXT間のBluetooth通信の接続設定
　(2)NXT間でのメッセージ送受信
　(3)スレーブ側NXTのコマンド起動
　(4)スレーブ側NXTのプログラム起動
　(5)NXT間でのより複雑な制御
　　【課題13】実行時のデバッグ対応支援プログラムの作成
　　【課題14】探索問題アルゴリズムへの挑戦
　Column10-1　トラブル・シューティング　以外と便利

Appendix

Appendix A　BricxCC機能概要一覧表　重要
Appendix B　標準CとNXCとの違い
Appendix C　NXC APIライブラリ関数一覧表
Appendix D　NXT拡張関連情報
Appendix E　NXTを使った教育カリキュラムの提案