KINARM

Creating Task Programs for Dexterit-E™

www.bkintechnologies.com

BKIN Technologies, Suite 1625, Biosciences Complex, 116 Barrie Street, Kingston, Ontario K7L 3N6 Canada
Tel: +1(888) 533-4393 Email: support@bkintechnologies.com

created by b·kin TECHNOLOGIES

Revision History

Revision	Description	Date	Approved By
1	Updated formatting style	5-Jul-2011	IEB

READ CAREFULLY BEFORE USING

END-USER LICENSE AGREEMENT FOR PRODUCTS MANUFACTURED AND/OR SOLD BY BKIN TECHNOLOGIES LTD.

IMPORTANT - READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or, if purchased or otherwise acquired by or for an entity, an entity) and BKIN Technologies Ltd., ("BKIN Technologies") for the BKIN Technologies equipment and/or software manufactured and/or sold by BKIN Technologies that accompanies this EULA, which may include associated media, printed materials, "online" or electronic documentation, proprietary protocols and databases, and Internet-based services ("Equipment" and/or "Software"). An amendment or addendum to this EULA may accompany the Equipment and/or Software. A copy of this EULA is available to you on the BKIN Technologies website (www.bkintechnologies.com) and available from BKIN Technologies if requested in writing. In the event of any errors, omissions or other discrepancies between this EULA, and any amendments or addendums accompanying this EULA, and the EULA made available by BKIN Technologies on its website, the terms of the latter shall control.

YOU AGREE TO BE BOUND BY THE TERMS OF THIS EULA BY INSTALLING OR IN ANY WAY USING BKIN TECHNOLOGIES EQUIPMENT AND/OR SOFTWARE. IF YOU DO NOT AGREE, DO NOT INSTALL OR IN ANY WAY USE THE EQUIPMENT AND/OR SOFTWARE; YOU MUST RETURN THE EQUIPMENT AND/OR SOFTWARE TO BKIN TECHNOLOGIES WITHIN TEN (10) DAYS. YOU MAY BE ELIGIBLE FOR A REFUND ACCORDING TO THE TERMS OF YOUR PURCHASE AGREEMENT.

1. LICENSE: BKIN Technologies grants you a single non-transferable and non-exclusive license to use the Equipment and/or Software subject to all the terms and conditions of this EULA. All rights not specifically granted to you by this EULA shall remain with BKIN Technologies. No rights are granted to you to sublicense, rent or market the Equipment and/or Software. No rights are granted to you under any patents, patent applications, trade secrets or other proprietary rights.

The Software accompanying this EULA is licensed by BKIN Technologies, but BKIN Technologies retains ownership of the Software itself, except for those parts which may be owned in whole, or in part by a third party. The rights granted under the terms of the EULA include any Software upgrades that replace and/or supplement the original Software, unless such upgrade contains a separate license. The provisions included in this EULA, including all reservations of rights and limitations of liabilities, extend to Software provided under this EULA by both BKIN Technologies and any third-party provider.

2. LICENSE RESTRICTIONS: ALL EQUIPMENT AND/OR SOFTWARE PROVIDED TO YOU UNDER THIS EULA SHALL BE USED SOLELY FOR RESEARCH PURPOSES AT YOUR INSTITUTIONAL FACILITIES ONLY. Equipment and/or Software is to be used only under your direction. You may install, use, access, display and run one copy of the Software on only a single computer, such as a workstation, except for any MDL files and M-files in source code format provided as part of the Software, which you may install, use, access, display and run on more than one computer as defined by The MathWorks, Inc. Software License Agreement. You are not authorized to install or use the Software on a network. You shall not provide access, directly or indirectly, to the Software via the Web or any Internet application, or any file-sharing method or system unless explicitly allowed in your purchase agreement. You may not rent, lease, or loan the Equipment and/or Software, use the Equipment and/or Software for supporting third party use of the Equipment and/or Software, time share the Equipment and/or Software or provide service bureau use.

You may not in any way clone, reverse engineer, decompile, or disassemble the Equipment and/or Software, except and only to the extent that such activity is expressly permitted by this EULA and by applicable law notwithstanding this limitation. You may modify any MDL files and M-files provided as part of the Software to derive other MDL and M-files for use with the Equipment and/or Software, however, any such derived works must be used in accordance with all terms and conditions of this EULA. You may use only the binary format of the Software provided under this EULA with the exception of any MDL files and M-files provided as part of the Software, which may be used in source code format as defined by The MathWorks, Inc. Software License Agreement.

You may make one copy of the Software for backup purposes, but all copyright and proprietary notices in the original must be included in such copy.

3. SOFTWARE MAINTENANCE SERVICE: During any paid Maintenance Service term, BKIN Technologies shall provide service for the Software consisting of: delivering subsequent releases of the Software, if any; exerting reasonable efforts to both (a) provide, within a reasonable time, workarounds for any material programming errors in the current release of the Software that are directly attributable to BKIN Technologies, and (b) correct such errors in the next available release, provided you provide BKIN Technologies with sufficient information to identify the errors (Maintenance Service). During this same paid Maintenance Service term, you shall also be entitled to receive technical support for the current release of the Software.

The fees for Maintenance Service for the one (1) year period commencing on the delivery date specified in the original purchase invoice ("Initial Maintenance Period") shall be included as part of software purchase price and no separate fee for

this Maintenance Service is payable by you to BKIN Technologies. After the Initial Maintenance Period, you may purchase Maintenance Service from BKIN Technologies, which shall be subject to an additional fee and mutual written agreement. BKIN Technologies shall have no obligation to provide Maintenance Service to you after the Initial Maintenance Period unless it is purchased by you in accordance with the agreement. Further, BKIN Technologies shall have the right to discontinue the use of any protocols and databases licensed on an annual basis if the Maintenance Service fee for said protocols and databases is not purchased.

4. CONFIDENTIALITY AND NON-DISCLOSURE: Provisions of this Article shall survive any termination of this EULA.

(A) Equipment and/or Software

You acknowledge that all Equipment and/or Software accompanying this EULA are proprietary products of BKIN Technologies or its suppliers. All Software shall remain the property of BKIN Technologies. You will not disclose or otherwise make available to any third party the Equipment and/or Software, any modified or derived works therefrom, or information contained therein, in any form, except to your employees and users for purposes limited to and specifically related to your use of the Equipment and/or Software in accordance with this EULA. You shall take appropriate action by instruction or signed agreements with such employees and users to satisfy your obligations under this EULA. If for any reason you or your employees or users gain access to BKIN Technologies materials containing any confidential or proprietary marking, or BKIN Technologies software source code to which you do not have a right of access under a written agreement between you and BKIN Technologies, you agree to not examine, use, copy, or keep such items, but shall return them promptly to BKIN Technologies. Your obligations of confidentiality and nondisclosure shall apply to all forms of Equipment and/or Software received.

(B) Injunctive Relief

Because harm not adequately compensable might result from unauthorized disclosure of proprietary or confidential information, BKIN Technologies may seek injunctive relief if you breach your obligations of confidentiality and nondisclosure under this EULA.

5. PATENT AND COPYRIGHT: All or parts of the Equipment and/or Software may have been patented or copyrighted by BKIN Technologies or third-party providers. Patent or copyright notices have been included in the Software for protective purposes and such notices shall not be construed as causing publication of the Software.

6. CHARGES: You agree to pay all charges arising from the license of the Equipment and/or Software. In addition, you agree to pay, or reimburse BKIN Technologies for paying, all taxes (except incomes taxes) and other government charges based on or measured by the charges set forth in this EULA, or based on the Equipment and/or Software, its use, and any services provided herein, now or hereafter imposed by any governmental authority, including but not limited to Provincial or State Sales Tax, Goods and Services Tax, Harmonized Sales Tax, and Use or Gross Receipt taxes.

7. LIMITED WARRANTY: BKIN Technologies warrants that all KINARM™ products will be substantially free from defects in material and workmanship. In the event of a breach of this warranty, provided that BKIN Technologies is notified of the defects within 12 months of shipment or before the expiry of any extended warranty purchased from BKIN Technologies, BKIN Technologies will exert reasonable efforts to correct the defects in materials and/or workmanship. These efforts will include repair or replacement of the defective part at customer's location or at BKIN Technologies, such determination at BKIN's discretion. BKIN will cover shipping costs by ground courier to and from customer's location should BKIN ask the customer to return the part to BKIN for repair.

BKIN Technologies warrants that all Software will conform in all material respects to its published specifications when operated on and with the Equipment specified. In the event of a breach of this warranty, provided that BKIN Technologies is notified of the defects within the Initial Maintenance Period, BKIN Technologies will, at its option, exert reasonable efforts to correct the defects, replace the Software, or terminate the EULA and refund any charges paid by you for the Software.

8. DISCLAIMER OF WARRANTIES, LIABILITIES, INDEMNIFICATION: The limited warranty that appears above is the only express warranty made to you and is provided in lieu of any other express warranties or similar obligations, if any, created by any advertising, documentation, packaging, or other communications.

BKIN TECHNOLOGIES MAKES NO OTHER WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. BKIN TECHNOLOGIES WILL IN NO EVENT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF THIS EQUIPMENT AND/OR SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. SPECIFICALLY, BKIN TECHNOLOGIES IS NOT LIABLE FOR ANY COSTS, SUCH AS LOST PROFITS OR REVENUE, LOSS OF EQUIPMENT, LOSS OF SOFTWARE, LOSS OF DATA, COSTS OF SUBSTITUTES, CLAIMS BY THIRD PARTIES, PERSONAL INJURY OR OTHERWISE, ARISING OUT OF OR RELATED TO YOUR USE OR INABILITY TO USE BKIN TECHNOLOGIES EQUIPMENT AND/OR SOFTWARE, HOWEVER CAUSED. BKIN TECHNOLOGIES DOES NOT WARRANT THAT THE OPERATION OF THE BKIN TECHNOLOGIES EQUIPMENT AND/OR SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE OR THAT DEFECTS WILL BE CORRECTED. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY BKIN TECHNOLOGIES OR A BKIN TECHNOLOGIES AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY BEYOND THAT SPECIFIED IN THIS EULA.

BKIN TECHNOLOGIES SHALL NOT BE LIABLE TO YOU FOR ANY CLAIM WHICH IS BASED UPON THE USE OF THE EQUIPMENT AND/OR SOFTWARE, OR ANY PART OF IT, IN CONNECTION WITH EQUIPMENT, SOFTWARE, OR DEVICES NOT FURNISHED BY BKIN TECHNOLOGIES, OR IN ANY MANNER FOR WHICH THE EQUIPMENT AND/OR SOFTWARE WAS NOT DESIGNED, OR IN ANY MANNER FOR WHICH THE EQUIPMENT AND/OR SOFTWARE HAS BEEN MODIFIED BY OR FOR YOU.

YOU ARE RESPONSIBLE FOR THE SELECTION OF EQUIPMENT AND/OR SOFTWARE TO ACHIEVE ITS INTENDED RESULTS, USE OF THE EQUIPMENT AND/OR SOFTWARE, AND THE RESULTS OBTAINED THEREFROM. YOU AGREE TO INDEMNIFY AND HOLD BKIN TECHNOLOGIES AND ITS SUPPLIERS HARMLESS WITH RESPECT TO ALL CLAIMS, INCLUDING THOSE BY THIRD PARTIES, ARISING OUT OF YOUR USE OF THE RESULTS OF OPERATION OF THE EQUIPMENT AND/OR SOFTWARE.

9. TERM: You shall have the right to use the Equipment and/or Software indefinitely, subject to the provisions of this EULA. You understand and agree that the Maintenance Service for any Software provided (which may include continuing access to protocols and databases) will terminate automatically upon expiration of the Initial Maintenance Period included with the purchase of the Software. Thereafter, the Maintenance Service term may be renewed at the then current price and for the then-applicable term, as long as BKIN Technologies offers such Maintenance Service, in accordance with the provisions of this EULA.

10. TERMINATION: This EULA is effective until terminated. Termination of this EULA shall not relieve you of any obligations under any invoiced fees. Upon termination of this EULA you must promptly return all Equipment to BKIN Technologies, and you must return all copies of the Software to BKIN Technologies or certify in writing to BKIN Technologies that all copies of the Software have been destroyed.

This EULA may be terminated automatically without notice from BKIN Technologies as follows:

(A) by you any time; or by either

(B.1) non-payment by you of any amount due under applicable invoices for the Equipment and/or Software, which non-payment continues for a period of ten (10) days; or

(B.2) non-performance by you of any other term or condition of this EULA, or of any other agreement between you and BKIN Technologies related to the Equipment and/or Software, subject to the provisions of this EULA.

11. ASSIGNMENT: You may not assign this EULA without the written consent of BKIN Technologies.

12. APPLICABLE LAW: This EULA shall be deemed to have been made in Ontario and is governed by, and construed and enforced in accordance with, the laws of the Province of Ontario, Canada, and the federal laws applicable therein. You and BKIN Technologies attorn to the non-exclusive venue and jurisdiction of the Courts of Ontario in respect of any matter or dispute arising from this EULA and its subject matter.

13. SURVIVAL OF AGREEMENTS: Notwithstanding the termination or completion of this EULA, all indemnities, warranties, restrictions and duties of confidentiality and non-disclosure in this EULA will continue in full force and effect to the extent required for their full observance and performance.

14. ENTIRE AGREEMENT; GOVERNING LANGUAGE: This EULA, including any amendments and addendums that may accompany it, constitute the entire agreement between you and BKIN Technologies related to the Equipment and Software and any related support services, and supersedes all prior or contemporaneous oral and written communications, proposals and representations of any kind with respect to the Equipment, Software, and any other subject matter covered by this EULA. The terms and conditions of any subsequent invoice, waiver, amendment or other such agreement used by BKIN Technologies in connection with this EULA shall be considered valid and enforceable to the extent that such terms and conditions can be interpreted as consistent with this EULA. If such terms and conditions cannot be interpreted as consistent with this EULA, the terms of this EULA shall control. If any provision of this EULA is held to be void, invalid, unenforceable or illegal, the other provisions shall continue in full force and effect.

Any translation of this License is done for local requirements and in the event of a dispute between the English and any non-English versions, the English version of this License shall govern.

1 Welcome

Welcome to Creating Task Programs for Dexterit-E™. This guide is intended for those users of BKIN Dexterit-E who wish to create customized Task Programs. This guide introduces:

- the concept of Task Programs
- the basic code structure of Task Programs
- numerous example Task Programs

This guide assumes that the end user is familiar with using BKIN Dexterit-E, including loading Task Programs, altering Task Protocols and the use of the Parameter Tables for the Task Protocols. It also assumes that the end user is familiar with Simulink and Stateflow.

1.1 Introduction to Custom Task Programs

One of the strengths of BKIN Dexterit-E is the flexibility provided the end-user in customizing the software to meet their own needs. As outlined in the Dexterit-E User Guide, BKIN Dexterit-E provides users the flexibility of defining multiple Task Protocols for a given task. The purpose of this guide is to describe the next level of flexibility: creating a custom Task Program, which BKIN Dexterit-E can then use for running a novel task.

Task Programs are small programs used by BKIN Dexterit-E that define and control system behavior during a single trial of a task. Task Programs are created to define system behavior for a general class of tasks. For example, a Task Program could define that during a trial a target will turn on, once a subject reaches to that target it will turn off and another target will turn on, once the subject reaches to the second target it will turn off and then the trial is over (i.e. the general class of point-to-point reaching tasks). The details of the task (e.g. the target location, color and number of trials) are not defined by the Task Program, but rather are specified as parameters in the Task Protocol and are defined through BKIN Dexterit-E's Windows-based user interface.

Coding a Task Program involves Mathworks' Simulink and Stateflow toolboxes. These toolboxes provide a graphical programming environment in which Task Programs are developed and include many prebuilt functions. BKIN Technologies provides with BKIN Dexterit-E a custom library of Simulink blocks that are specific to Task Programs and greatly assist in rapid code development. For further information on Stateflow and/or Simulink, please read the Mathworks' introduction to each of these toolboxes (http://www.mathworks.com/access/helpdesk/help/helpdesk.html).

2 Software Setup

This chapter provides a description of the software setup required to create Task Programs. Task Programs for BKIN Dexterit-E are created within the Simulink environment within Matlab. For an end-user to create or customize a Task Program, Matlab, various Matlab toolboxes and a C/C++ compiler must all be installed on a computer running Windows XP. This computer does not need to be the computer running BKIN Dexterit-E. In addition, the end-user will need to install a set of Simulink libraries provided with BKIN Dexterit-E called the BKIN Dexterit-E Task Development Kit (*TDK*). While the TDK is included as part of BKIN Dexterit-E, Matlab and its associated toolboxes are not included, nor is a C/C++ compiler.

2.1 BKIN Dexterit-E and Task Development Kit Version Compatibility

Due to changes in the underlying structure of the software, there are compatibility issues between different versions of BKIN Dexterit-E and the Task Development Kit with which Task Programs are built. Task Programs must be built with a TDK version compatible with the BKIN Dexterit-E version to be used. Refer to subsequent sections on issues for updating a Task Program from one version of the TDK to another.

Table 2-1: BKIN Dexterit-E and TDK Version Compatibility

BKIN Dexterit-E Version	Compatible TDK Versions
2.0.x	1.0.x
2.1.x	1.1.x
2.2.0	1.2.0
2.3.n	2.3.m (where m ≤ n)
3.0.n	3.0.m (where m ≤ n)
3.1.n	3.1.m (where m ≤ n)

Note:Versions of the BKIN Task Development Kit distributed with BKIN Dexterit-E 2.2 and earlier were known as the Dexterit-E Simulink Library, or "Simlib".

2.2 BKIN Dexterit-E, Matlab and C/C++ Compiler Version Compatibility

BKIN Technologies verifies and supports its products for use with only specific combinations of versions of BKIN Dexterit-E, Matlab and the C/C++ compilers supported by Matlab. The table below shows which versions of Matlab and C/C++ compilers have been verified to work with BKIN Dexterit-E. Untested combinations of versions may work, however this is not guaranteed.

Table 2-2: BKIN Dexterit-E, Matlab and C/C++ Compiler Version Compatibility

Matlab Release	Supported C/C++ Compilers	Supported BKIN Dexterit-E Versions
2007b (*)	Microsoft Visual C++ 6.0 (SP6) Microsoft Visual C++ 2005 Express Edition Open Watcom 1.3	2.2.x 2.3.x 3.0.x 3.1.x
2010a**	Microsoft Visual C++ 2008 Express Edition Open Watcom 1.8	3.0.x 3.1.x

* These versions of Matlab require a patch to work properly. See Section 2.4 - "Matlab, Simulink and xPC Target" for more details.

** The NI PCI-6229 card in systems made from 2010 on is only supported with Matlab 2010a

2.3 C/C++ Compiler

Task Programs are built by Matlab and the xPC Target toolbox using a third party C/C++ compiler. A version of the chosen C/C++ compiler must be installed on the same computer as Matlab for task development. This task development computer can be, but does not need to be, the computer running BKIN Dexterit-E.

Each release of Matlab and the toolboxes used to develop Task Programs support specific versions of freely-available compilers such as Microsoft Visual C++ Express and Open Watcom. Consult the Supported and Compatible Compilers list in the documentation for your Matlab release to determine which compilers are supported with that release. Using an unsupported compiler will have undefined results.

2.4 Matlab, Simulink and xPC Target

In order to create custom Task Programs for use with BKIN Dexterit-E, the end-user must install Matlab R2007b or R2010a, along with the following Matlab toolboxes:

- Simulink
- Stateflow
- Stateflow Coder
- Real-Time Workshop
- xPC Target

Matlab R2007b

If using Matlab R2007b, the end-user must apply the following patch to fix a Matlab bug.

1. Back up the all of the `*.rtb` files in `<$MATLAB>\toolbox\rtw\targets\xpc\target\kernel\*.rtb` (where `$MATLAB` is your root MATLAB directory as returned by the `MATLABROOT` command).
2. Back up the `xpcapi.dll` file in `<$MATLAB>\toolbox\rtw\targets\xpc\xpc\private\xpcapi.dll`.
3. Download patched versions of these files from the BKIN Technologies website. From the main page at `http://www.bkintechnologies.com/` choose Resources, and then on the left side choose Downloads. Log in using your username and password. On the left side choose Downloads → Current. Download `Patch_R2007b` from the Matlab section. This download contains patched versions of the files backed up in steps 1 and 2.
4. Replace all of the `*.rtb` files with the patched versions.
5. Replace the `xpcapi.dll` file with the patched version.
6. If an xPC Target boot disk was created using the original R2007b, then a new boot disk must be created after the patched files are installed.

Setting Matlab to Use the Correct C Compiler

For Matlab to use the appropriate C compiler, the end-user must set up the C compiler within Matlab:

1. At the Matlab prompt, type `xpcexplr`.
2. In the xPC explorer dialog change the compiler path to whichever path your MS Visual C++ is installed in. (e.g. `C:\Program Files\Microsoft Visual Studio`)

2.5 Custom Task Program Development Kit (TDK)

Simulink libraries provided with BKIN Dexterit-E are installed by default in `C:\Program Files\BKIN Technologies\BKIN Task Development Kit x.x`, where x.x is the version number of the BKIN Dexterit-E Task Development Kit. These libraries need to be installed on the computer that will be used to create and compile custom Task Programs, (i.e. the same computer that will need Matlab, various Matlab toolboxes and a C/C++ compiler, as described below). For these libraries to be available within Simulink, the Matlab path needs to be changed as per the following instructions:

1. From the Matlab File menu, choose Set Path.
2. Remove entries for any previous version of the TDK that exist in Matlab path. For example, if the previous version was 1.2 with the PMAC-PCI, the following paths would need to be highlighted and then the Remove button clicked:
   a. `C:\Program Files\BKIN\Simlib 1.2`
   b. `C:\Program Files\BKIN\Simlib 1.2\PMAC-PCI`

   Note: Versions of the BKIN Task Development Kit distributed with BKIN Dexterit-E 2.2 and earlier were known as the Dexterit-E Simulink Library, or "Simlib". These libraries were installed by default in C:\Program Files\BKIN\Simlib x.x, where x.x is the version of the library.
3. Click Add Folder… and add the new TDK path:
   a. `C:\Program Files\BKIN Technologies\BKIN Task Development Kit x.x`
4. Click Add Folder… and add the new TDK motion control card path:
   a. If you have a PCI version of the PMAC motion control card, add: `C:\Program Files\BKIN Technologies\BKIN Task Development Kit x.x\PMAC-PCI`
   b. If you have an ISA version of the PMAC motion control card, add: `C:\Program Files\BKIN Technologies\BKIN Task Development Kit x.x\PMAC-ISA`

   Note: All KINARM Labs sold after January 2003 contain PCI cards. KINARM Exoskeleton Labs sold prior to this date contain ISA cards.
5. Double-check that only one version of the TDK is in the Matlab path.
6. Double-check that only one TDK motion control card path is in the Matlab path.
7. Click Save.
8. Click Close.
9. Restart Matlab.

Steps 5-9 of these instructions are necessary because of the behavior of Simulink and Real-Time Workshop. Real-Time Workshop does not use the order of directories in the Matlab path for identifying which file to use during the build process, so it is critical that duplicate files are not located in the Matlab path. Simulink does not use changes made to the Matlab path until Matlab is restarted, so it is also critical to restart Matlab after changing the Matlab path.

NI PCI-6071E

Human KINARM Labs from before 2011 include a National Instruments (NI) PCI-6071E card. The exception are those systems that use an external data acquisition system such as Plexon. If your hardware system uses NI's PCI-6071E DAQ card, you may want to use the `adnipcie.c` file supplied with the TDK (in the directory `C:\Program Files\BKIN Technologies\BKIN Task Development Kit x.x`). Reasons for doing so are to increase the sample time of the analog inputs on the card, to ensure that you do not get cross-talk between channels. See Section 6.6, “Error CPU overload” of this guide for more information related to this version of the file. To ensure that this file is used, rename the `adnipcie.c` and `adnipcie.mex32` files supplied with your version of Matlab to something else (e.g. `adnipcie_OLD.c` and `adnipcie_OLD.mex32`). These files can be found in the directory `$MATLAB\toolbox\rtw\targets\xpc\target\build\xpcblocks`. `$MATLAB` is the root Matlab directory as returned by entering `matlabroot` in the MATLAB command window.

NI PCI-6229

Human KINARM Labs from 2011 forward include a National Instruments (NI) PCI-6229 card. If you plan to collect the data from the analog channels on the card you are encouraged to use the new “Autodetected National Instr. PCI-6071E / PCI-6229” block in the “KINARM I/O” library. This will make your task portable between older and newer KINARM models. You will need to use Matlab 2010a to compile your models if you wish to use the NI PCI-6229.

3 From Concept to Code: Converting an Idea into a Task Program

This chapter provides an overview of creating a Task Program from the descriptive outline of a task. It begins with a description of what a Task Program is, followed by the settings and configuration parameters necessary within Simulink, and then a walk-through example of going from a descriptive outline of a task to actual code that can be used by BKIN Dexterit-E.

In order to properly understand this chapter and the remainder of this guide, the end-user must be familiar with using BKIN Dexterit-E. In particular how to load, manipulate and design Task Protocols for an existing Task Program. This guide assumes that the end user is familiar with BKIN Dexterit-E and these concepts.

3.1 Introduction to a Task Program

Task Programs are small programs used by BKIN Dexterit-E that define and control the system behavior that can occur during a single trial of a task. For example, a Task Program could define:

1. During a trial a target will turn on.
2. Once a subject reaches to that target it will turn off and a second target will turn on.
3. Once the subject reaches to the second target it will turn off.
4. The trial is over.

In this example, note that the Task Program does not define certain parameters, such as the location, size and color of the two targets. Nor does it define how many trials will occur or how these parameters change from trial to trial. These parameters are part of what is called a Task Protocol. Together, the Task Protocol and Task Program define everything that will occur during a task. Thus a more explicit definition of a Task Program is that it defines system behavior for a general class of tasks, this example being a general class of point-to-point reaching task. The separation of Task Program from the parameters stored in a Task Protocol means that multiple Task Protocols can be associated with a single Task Program.

3.2 The Programming Environment: Simulink and Stateflow

Programming a new Task Program involves Mathworks' Simulink and Stateflow toolboxes. Simulink is the graphical programming environment in which Task Programs are developed and is a graphical representation of data flow in the task. Stateflow is a graphical design tool for developing event-driven state machines called Stateflow Charts. A Stateflow Chart is a form of flowchart in which independent system states are created and events are defined to allow transitions between those system states. Simulink comes with many of its own pre-made functions, or blocks. BKIN Technologies provides its own library of Simulink blocks (referred to as the BKIN Task Development Kit, or "TDK") that are specific to BKIN Dexterit-E's Task Programs and assist in rapid code development. Simulink and Stateflow were chosen to provide a stable programming environment with as much flexibility for the end-user as possible.

If you are unfamiliar with Stateflow and/or Simulink, we recommend going to the Mathworks' website (`http://www.mathworks.com/access/helpdesk/help/helpdesk.html`) or choose Help→ Product Help from the Matlab menu, and learn more about Simulink and Stateflow. Understanding Simulink and Stateflow is essential to understand the code for the Task Programs shown below and this user guide has been written with the assumption that the end-user has a basic knowledge of Stateflow and Simulink. For questions related to which version of Matlab to use, and therefore which version of Simulink/Stateflow, please refer to Section 2.2 - "BKIN Dexterit-E, Matlab and C/C++ Compiler Version Compatibility". In particular, the sections listed below are of particular significance.

Suggested Simulink Help Sections to Read

- Getting Started
- Simulink Basics
- Creating a Model
- Using the Embedded MATLAB Function Block
- Chapter summary for all other chapters (to be aware of other possible chapters of interest)
- Blocks - By Category/Alphabetical List (for reference)
- Examples (for reference)

Suggested Stateflow Help Sections to Read

- Getting Started
- Stateflow Concepts
- Stateflow Notation
- Stateflow Semantics
- Creating Stateflow Charts
- Extending Stateflow Chart Diagrams
- Defining Events and Data
- Using Actions in Stateflow
- Semantic Rules Summary
- Chapter summary for all other chapters (to be aware of other possible chapters of interest)
- Examples (for reference)

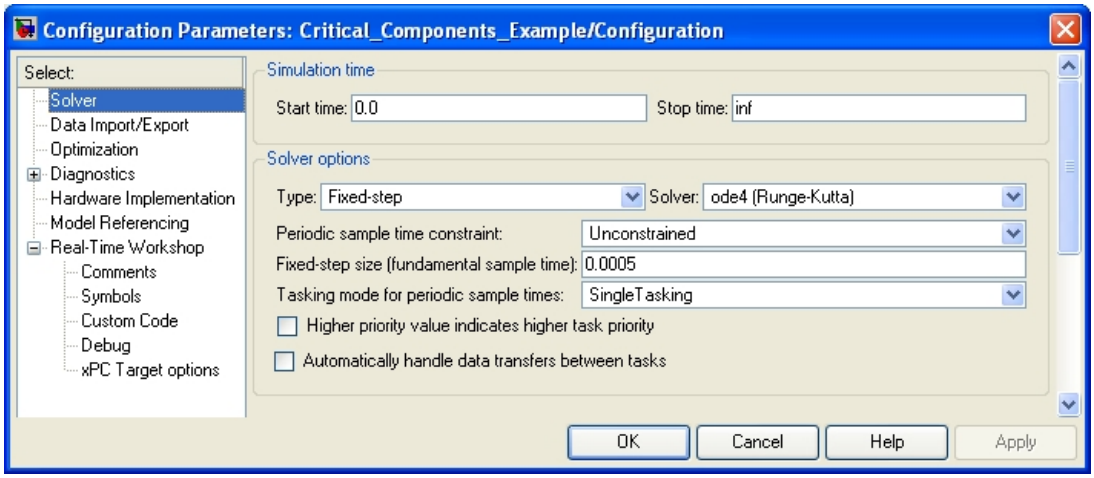
The above suggestions are a minimum suggested set of reading. Which other sections should be read will be user and need dependent.

3.3 Initial Configuration of `<your_task>.mdl`

In order for a Simulink model file to be compatible as a Task Program for BKIN Dexterit-E, various parameters need to be set in the Simulink model file. For users who are creating their own Task Program, we recommend starting with an existing `<Task Program>.mdl` and modifying it to suit their own needs. Examples of Task Program code can be downloaded from `www.bkintechnologies.com`. This approach will ensure that various settings in `<your_task>.mdl` are set correctly. You can then remove those blocks from the Task Program that are not needed, and add in other blocks as needed.

Alternatively a new Simulink model file can be used, with the parameters set as described below. To set the required parameters for a new Simulink model file, from within Simulink select Simulation→ Configuration Parameters and set the parameters to match those in the dialogs shown in **Figure 3-1**.

Fig 3-1. Configuration Parameters



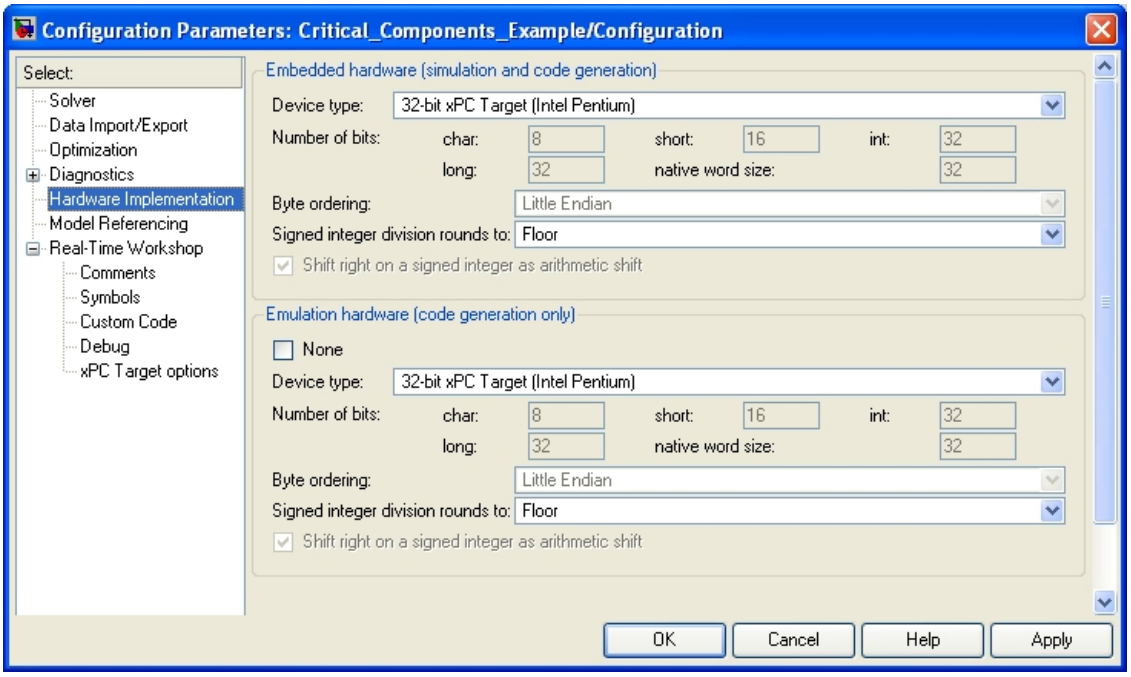
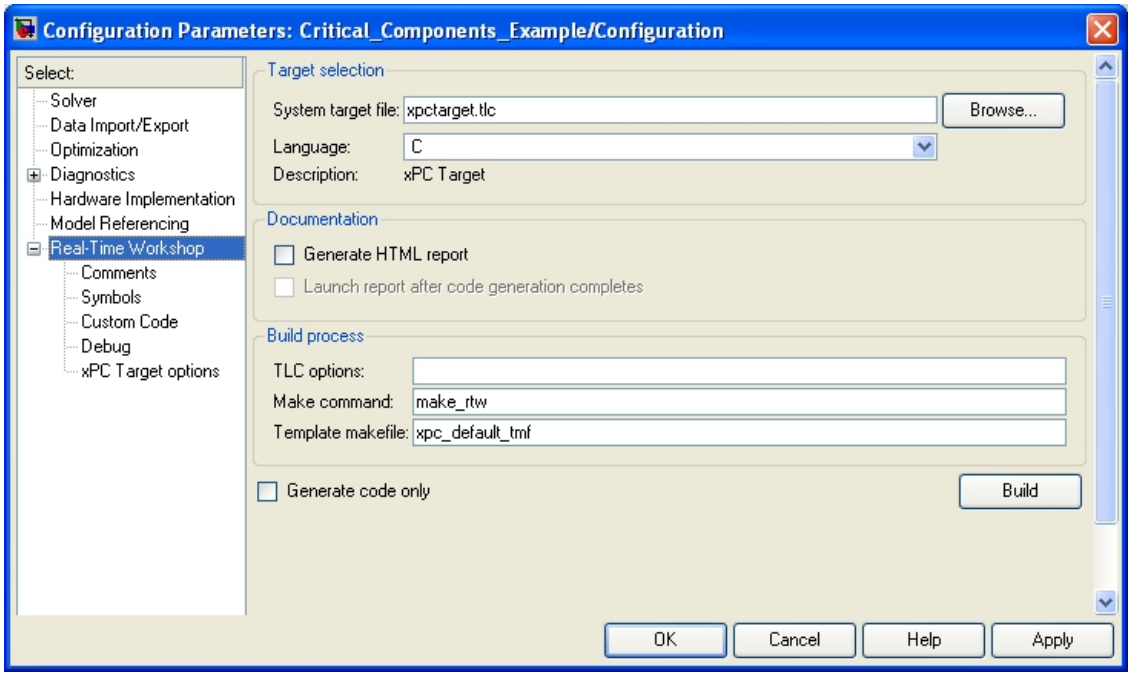
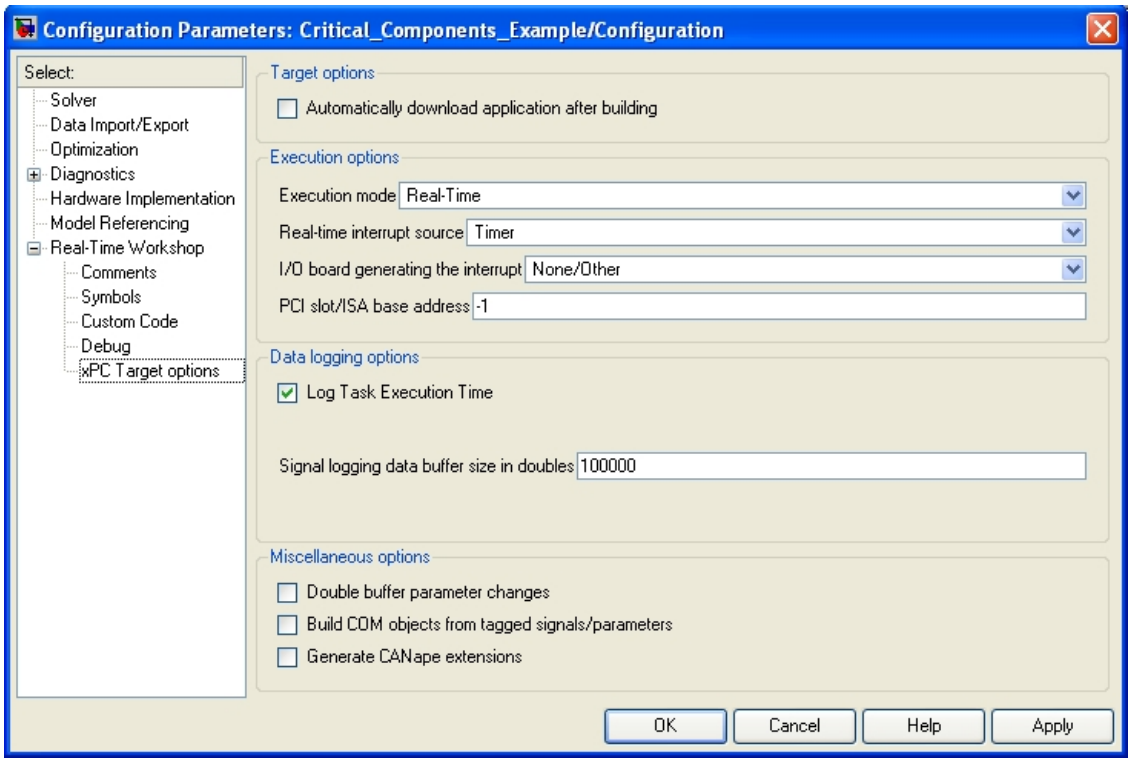


Figure 3-1: (cont.) Configuration Parameters



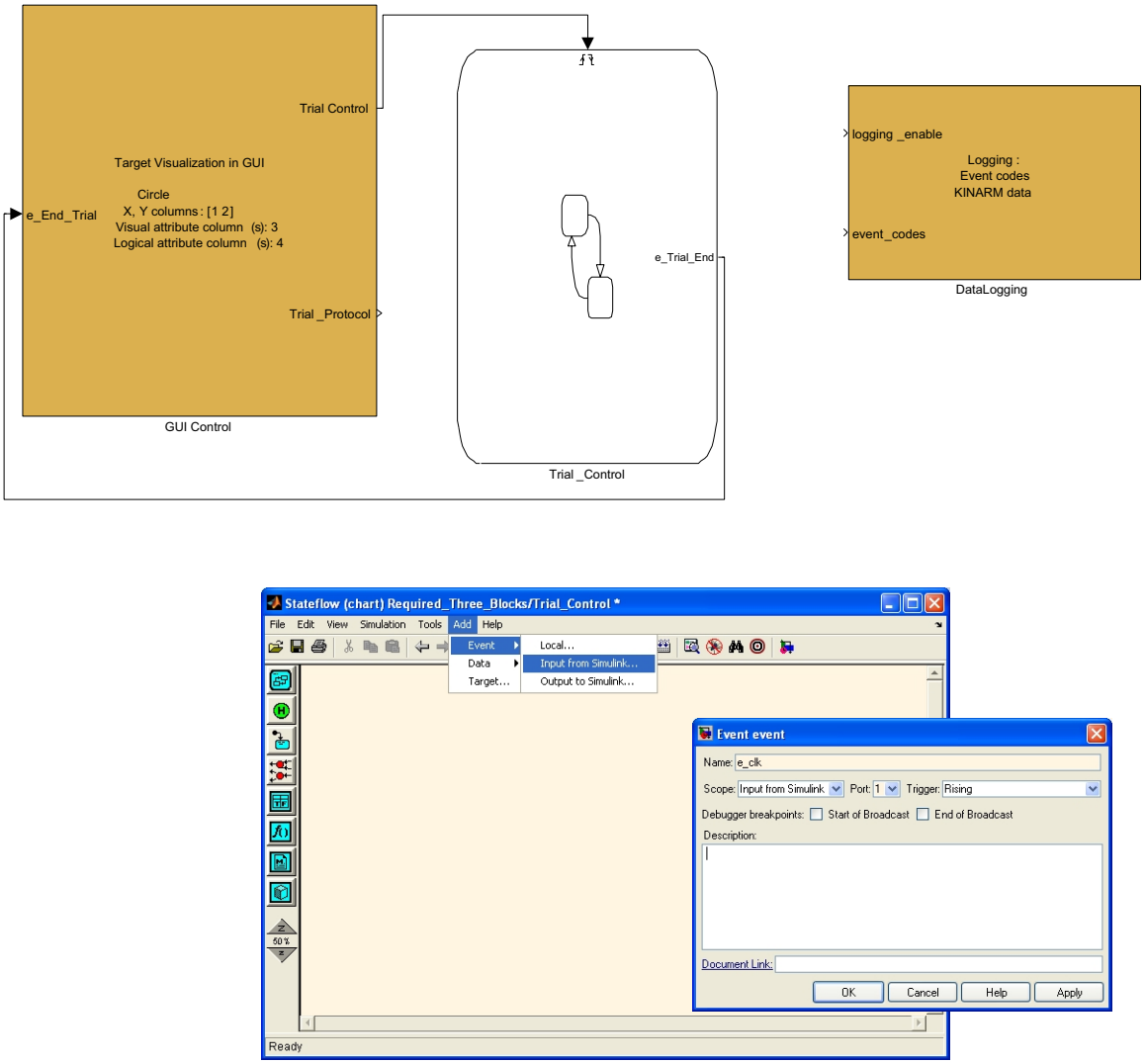


3.4 Required Simulink Blocks of `<your_task>.mdl`

After the Simulink parameters described in Section 3.3 - "Initial Configuration of <your_task>.mdl" have been set in `<your_task>.mdl`, three Simulink blocks required by BKIN Dexterit-E need to be added.

These blocks must all be present in the top level of the Simulink model as shown in **Figure 3-2**. The `GUI Control` and `DataLogging` blocks are part of the BKIN Task Development Kit, whereas the `Trial_Control` block is a Stateflow Chart and can be found in the Stateflow library.

Fig 3-2. Required Simulink Blocks for a Task Program



On the left of the Simulink diagram is the `GUI_Control` block which takes care of much of the communication between the Task Program (which runs on a real-time computer) and the BKIN Dexterit-E GUI (which runs on a separate, Windows-based computer). The `GUI_Control` block serves a number of important purposes. From a programmer's perspective, its most important function is to control the timing and sequence of trials and to output the appropriate Trial Protocol for each trial. It receives feedback from the Stateflow chart in the form of an `e_End_Trial event`, as well as an optional `Repeat_Trial` input (not used in this example).

In the center is a Stateflow chart (`Trial_Control`). This Stateflow chart controls the behavior of the system during a single trial. Various inputs provide events and data which can be used to drive the Stateflow chart and various outputs from the Stateflow chart can be used to drive external events. Much of the effort

in creating a Task Program will be programming the contents of the Stateflow chart, as described subsequently.

The `DataLogging` block logs all data to be saved by the Task Program, including KINARM-related data (i.e. position, velocity, etc.) as well as events and analog input data (not shown here). Data logging only occurs when the `logging_enable` input is set equal to 1 and the trial is running (i.e. data logging pauses when the trial is paused).

Initially, the `Trial_Control` Stateflow Chart will not have any inputs or outputs. To add the required inputs and outputs after dragging a new Stateflow Chart from the Stateflow library into the Task Program:

1. Open the Stateflow Chart by double-clicking on it in Simulink.
2. As shown in **Figure 3-2**, choose Add→ Event→ Input from Simulink...
3. In the new Event dialog, set the Name, Scope, Port and Trigger as shown in **Figure 3-2**.
4. Repeat steps 2-3 for a second input event, but set:
   a. Name: e_ExitTrialNow
   b. Scope: Input
   c. Port: 2
   d. Trigger: Either
5. Repeat steps 2-3 for a new output event and set:
   a. Name: e_Trial_End
   b. Scope: Output
   c. Port: 1
   d. Trigger: Either

For more information on creating and editing inputs and outputs to a Stateflow Chart, refer to Section 3.8 - "The Model Explorer - Adding Input and Outputs to a Stateflow Chart".

After these three input and output events have been created, the two signal wires shown between `GUI Control` and `Trial_Control` must be added. The first signal, from the `Trial_Control` output in the `GUI Control` block, must be wired into the top input in the `Trial_Control` block. The second signal must go from `e_Trial_End` event output in `Trial_Control` and be wired into `e_End_Trial` input in the `GUI Control` block.

There are almost always numerous other Simulink blocks present in the final Task Program, however, which ones are present will be different for different Task Programs. Section 3.7 - "Which Simulink Blocks to add?" provides more information on this issue, as does Chapter 6 - "Examples of Task Program Code". For more information on the functionality of individual Simulink blocks available in the BKIN Dexterit-E library, please click Help for each block from within Simulink.

3.5 A Basic Task Program

Creating a Task Program starts with a descriptive outline of a task. There needs to be a description of what kinds of things will happen during the task, when they will happen, if they are conditional and if so, conditional upon what. For example, details that need to be specified include anything that will affect or interact with the subject (e.g. a target will be presented or a load applied), or alternatively something that might be required for subsequent data analysis by the user (e.g. a record of an event when the subject presses a button). Typically the description needs to include all the variations of what can happen (e.g. sometimes there will be one target shown, sometimes two targets shown) and when and how these

variations will occur (e.g. one target for first 10 trials, two targets for next 10 trials). Often these descriptions are purely text-based, although they can include figures or flow charts.

As an example to be used throughout this chapter, we will describe here a sample 'center-out' reaching task. The rest of this chapter will use this example as a basis for going from concept to code.

Sample Task Description

In this sample task, a subject will be presented with a center target followed by a random selection of one of 8 peripheral targets. The subject will be instructed to reach to the first target and hold at that target. After holding at the target for a specified period of time, the target will turn off and a peripheral target will appear; the subject is to reach quickly and accurately to the second target. Once the peripheral target is reached, there is a delay before the target turns off and another delay until the start of the next trial. There will be 10 blocks of 8 trials, each block consisting one reach to each of the peripheral targets, presented in a random order.

3.6 Sample Task Breakdown: Task Program or Task Protocol?

Before the transformation from description to code can take place, we need to clarify what part of the task description goes into the Task Program, and what goes into the Task Protocol. Because changes in the Task Program require rebuilding and redeploying the Task Program, it is advantageous to minimize the number of times that process will have to take place. Choosing those parameters that are most likely to be changed from one variation of a task to another to be defined in the Task Protocol versus the Task Program will help achieve this goal. As described previously, a Task Program defines and controls the system behavior that can occur during a single trial of a task, so clarifying the definition of what constitutes a trial is the first step in the task breakdown. This step is then followed by choosing which parameters go into the Task Protocol and which are hard-coded into the Task Program.

Trial Definition

A typical task is composed of a defined number of trials. Sometimes each trial is identical, but typically there are predefined changes in task behavior from trial to trial. The key part of this step is choosing what portion of the task is being repeated over and over again. Often the choice is obvious, but that is not necessarily the case. In the center-out reaching task described above, one possible choice for what constitutes a trial includes the following sequence of events:

1. Presentation of a center-target
2. Subject reaches to the center target
3. Presentation of a peripheral target (after the required hold period at first target)
4. Subject reaches to the peripheral target
5. Pause before turning off second target

Choice of Task Protocol Parameters

Task Protocol parameters are set in BKIN Dexterit-E using the Task Protocol Parameter Tables. Their values are not defined in the Task Program. However, because they are used by the Task Program, we need to create variables for them for the Task Program. The choice of what to make a Task Protocol parameter needs to be made at this juncture. As mentioned above, having a parameter be set in the Task Protocol versus having it hard-coded in the Task Program provides a significant advantage: it allows us to quickly change task behavior without having to rebuild the Task Program.

A Task Program only defines what will happen during a single trial. Any parameter that changes from trial to trial should be part of the Task Protocol. In addition, it is useful to consider which parameters might be

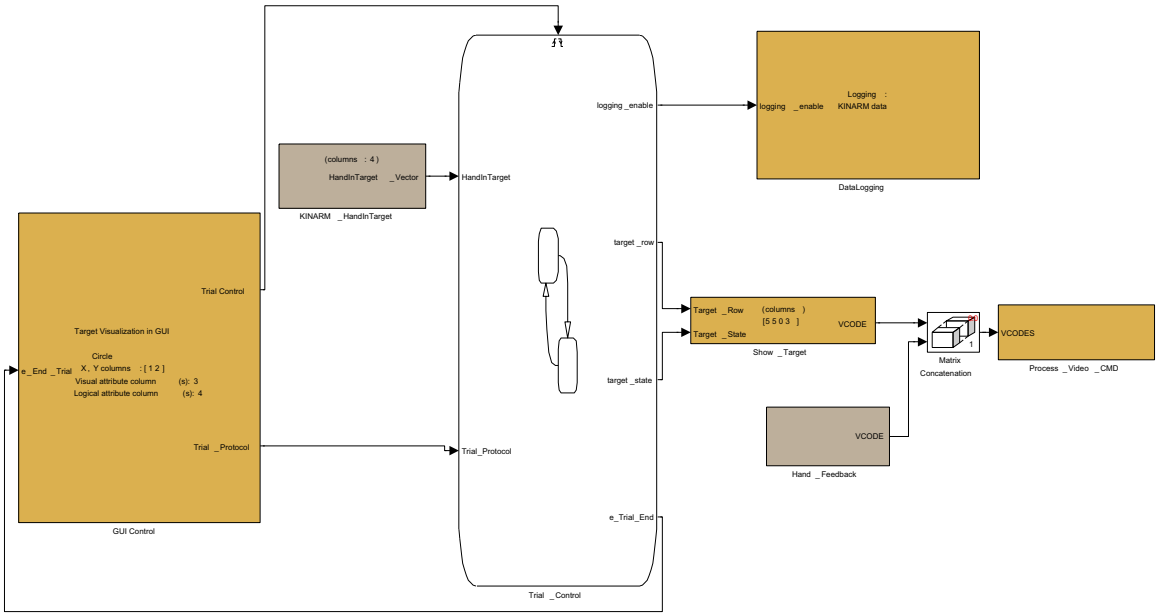
different in a future experiment. Any parameter that has a reasonable likelihood of changing its value can, and probably should, be set as a Task Protocol parameter.

For this example, the size, color and location of both the center and peripheral targets will be set as Task Protocol parameters. In addition, the length of time required to hold at the center target will be a parameter. A more complicated task will often be many more Task Protocol parameters. For example, if a Task Program included forces or loads then parameters defining the forces or loads would typically be set as Task Protocol parameters. In our Basic Task Program example, it would be possible to hard-code the size, color and location of the center target, because it is constant for all trials. However, by choosing it to be a Task Protocol parameter, we ensure that future variations of this task can include different target sizes, colors and locations, without having to rebuild the Task Program.

3.7 Which Simulink Blocks to add?

Assuming that the model has been configured correctly, as per Section 3.3 - "Initial Configuration of <your_task>.mdl", and that it contains the required Simulink blocks as per Section 3.4 - "Required Simulink Blocks of <your_task>.mdl", we can now add other blocks as necessary for this task. As shown below, in a screenshot of the final Simulink code for this task, there are numerous colored blocks, all of which are included as part of the BKIN Task Development Kit: gold-colored blocks, which are blocks specific to BKIN Dexterit-E, and brown blocks, which are blocks specific to KINARM. Each of these blocks is described briefly after the figure. For more information on any of these blocks, please refer to the examples in Chapter 6 - "Examples of Task Program Code", or right-click on any Simulink block and select Help.

Fig 3-3. Simulink Code for Example 3.5 - "A Basic Task Program"



The GUI Control, Trial_Control and DataLogging blocks were described in Section 3.4 - "Required Simulink Blocks of <your_task>.mdl". The Trial_Control block now has additional inputs and outputs. The inputs are used to drive the Stateflow Chart within the Trial_Control block, and the outputs are used to drive the other Simulink blocks. How to create these inputs and outputs is shown in Section 3.8 - "The Model Explorer - Adding Input and Outputs to a Stateflow Chart".

The `KINARM_HandInTarget` block provides feedback about which targets the hand is in. This block allows the Stateflow chart to act on that information as will be shown below.

Visualization of targets and hand feedback is controlled by the remaining three colored blocks: `Show_Target`, `Hand_Feedback` and `Process_Video_CMD`. The `Show_Target` block creates a `VCODE`, which is a visual command to be processed by the `Process_Video_CMD` block to draw a target. The `Hand_Feedback` block creates a `VCODE` for the hand feedback to be shown to the subject. The `Process_Video_CMD` processes the video commands and outputs them in an appropriate format to the computer controlling the video.

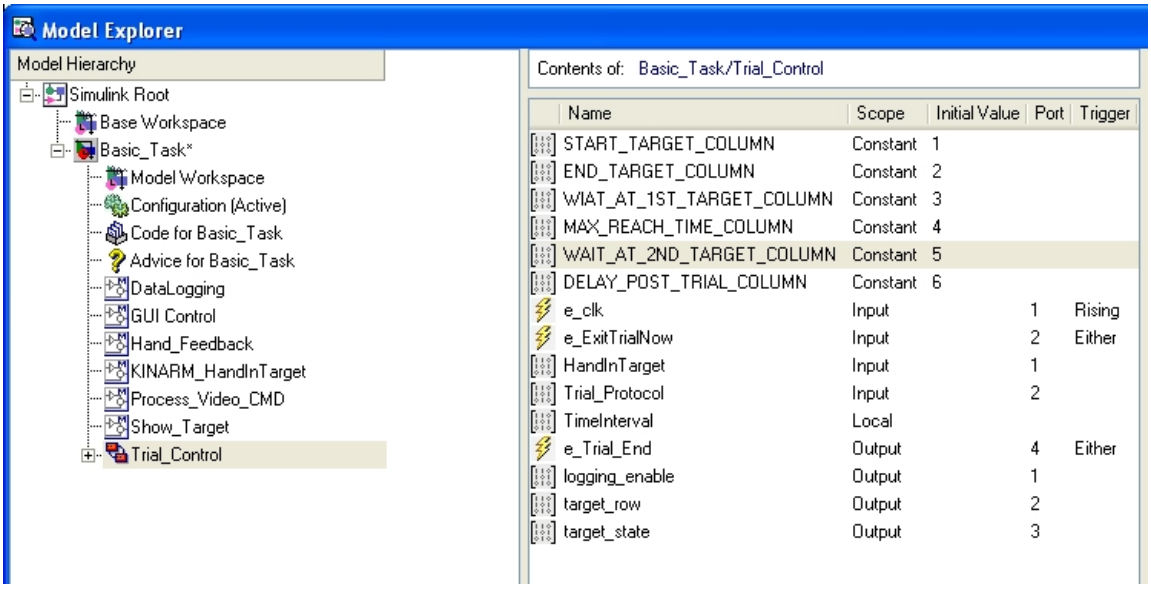
For more detailed information on each of these blocks, open up the relevant block within Simulink and click the Help button.

Note: The GUI Control block, DataLogging block, and Trial Control Stateflow chart must all be in the "top-level" of the Task Program, as shown here.

3.8 The Model Explorer - Adding Input and Outputs to a Stateflow Chart

There are two ways to add inputs and outputs to a Stateflow Chart. As shown in **Figure 3-2** and described in Section 3.4 - "Required Simulink Blocks of <your_task>.mdl", one can use the Stateflow Editor (i.e. the Add menu option). Alternatively, inputs and outputs can be defined using the Model Explorer. The advantage of understanding the Model Explorer is that it presents all previously defined inputs and outputs, allowing them to be modified. From within Simulink or Stateflow, choose View → Model Explorer to open it up. A simplified screenshot of the Model Explorer for this example task is shown below.

Fig 3-4. Model Explorer for Example 3.5 - "A Basic Task Program"



In the left-hand pane is the model hierarchy, in which the Stateflow Chart `Trial_Control` has been selected. In the right-hand pane are the inputs, outputs, constants and local variables that have been defined for the `Trial_Control` Stateflow chart. There are two types of inputs, outputs and local variables: data (matrix symbol) and events (lightning symbol). Data contain numerical values for reference in the Stateflow diagram (i.e. floating point or integer values, as defined), whereas events drive the Stateflow diagram execution (see Section 3.2 - "The Programming Environment: Simulink and Stateflow" for

suggested reading if the concept of an event is not clear). For more information on how to use the Model Explorer, please refer to the Simulink Help documentation.

Whenever an input or output is defined for a Stateflow Chart in the Model explorer, it has an effect on the Simulink block containing the Stateflow Chart. Data inputs appear as input ports on the left side of the Stateflow Chart block (e.g. see HandInTarget input of Trial_Control, in **Figure 3-3**). Data outputs appear as output ports on the right side of the Stateflow Chart block (e.g. see logging_enable output of Trial_Control, in **Figure 3-3**). Input events must be wired to a single input port at the top of the Stateflow Chart block (e.g. see signal from `Trial Control` output of `GUI Control` block in **Figure 3-3**); all input events share the same input port. Output events appear on the right side of the Stateflow Chart as output ports, and are listed after the Data output ports (e.g. see `e_Trial_End` output of `GUI Control` block in **Figure 3-3**).

The input and output events listed in **Figure 3-4** must be defined for all Stateflow Charts in a Task Program, and must be wired as shown in Section 3.4 - "Required Simulink Blocks of <your_task>.mdl". Creating these required input and output events using the Stateflow Editor has already been described in Section 3.4 - "Required Simulink Blocks of <your_task>.mdl". The meaning and use of the events is described here.

e_clk – This input event must be on port 1, and have a defined trigger of Rising. This event occurs every 1.0 ms, as long as the Task Program is running (i.e. as long as the task has started and is not paused). It originates in the `GUI Control` block.

e_ExitTrialNow – This input event must be the on port 2, and have a defined trigger of Either. This event occurs whenever a user clicks Pause and BKIN Dexterit-E has been set to pause the trial immediately. This event can be used by the end user to define what will happen in the Task Program when this event occurs. It originates in the `GUI Control` block.

e_Trial_End – This output event can be on any output event port, but must have a defined trigger of Either. This event is used by the end user to signal when a trial is over, so that the Task Program can update itself for the next trial. It terminates in the `GUI Control` block.

Note: All inputs, outputs, constants and variables to be used in a Stateflow chart, including both data and events, must be defined in the Model Explorer. If other input events are muxed with the GUI Control block events (i.e. e_clk and e_ExitTrialNow), the GUI Control block events must remain the first two signals of the muxed signal.

3.9 Sample Task: From Text Description to Stateflow Chart

In this section, the text description of what will happen during a single trial of this task will be converted to Stateflow code. This process will be done in several steps to help clarify the different ideas of what needs to happen for this transformation.

Text Description of Trial

Based on the description of the task and our subsequent definition of a trial for this task, we have parsed out a generic description of a trial for this task. It is important to recognize in the following description that no specific reference to shape, size or color of targets has been made, nor any reference to number of trials.

During a single trial of this task, a target will be presented to which the subject must move. Once the subject reaches the target, a specified time period will pass after which the target will disappear and a peripheral target will appear. Once the subject reaches to the peripheral target, a specified time period later the target will disappear and the trial is over.

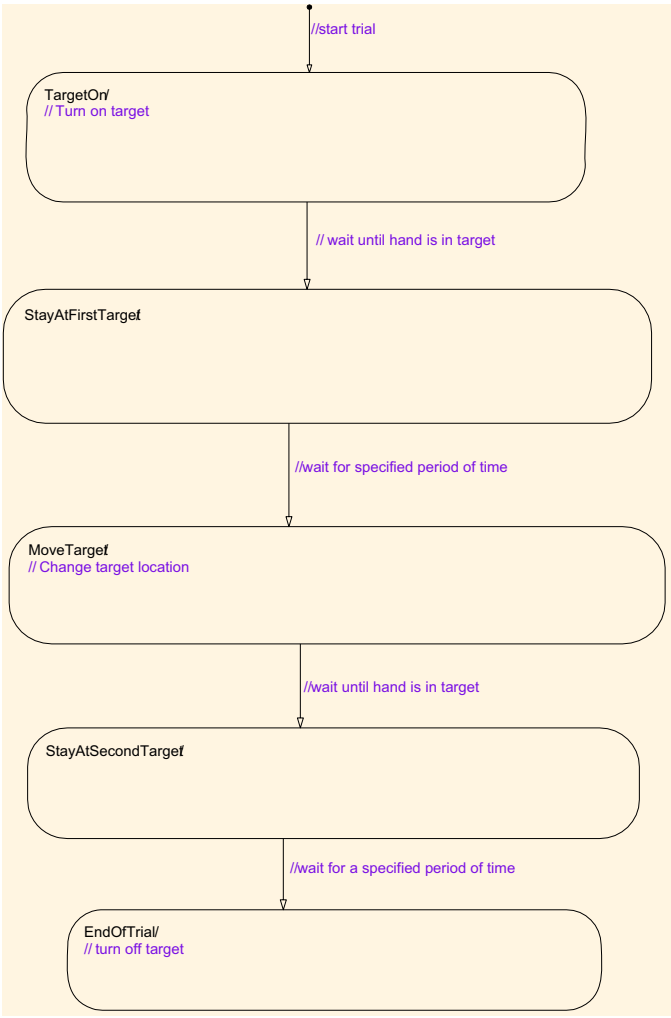
Note: No specific reference to shape, size or color of targets has been made, nor any reference to number of trials, etc.

Conversion to Simplified Flow Chart

The figure below shows a flow chart version of what was just described in text of what will happen during a single trial for this task. In this flow chart, the diagrammatic conventions of Stateflow have been used. States are represented by ovals, transitions between states are represented with arrows, and small circles known as 'connective junctions' provide a way to connect multiple transitions together between states. In Stateflow, the system can generally only exist in one state at a time, and the system must be in one of the defined states (i.e. the system cannot enter and stay at a connective junction). Changes from one state to another can only occur if there is a transition between the states and the conditions required for that transition are true.

In this step of the conversion process, we are not yet going to introduce any code. Instead, each state includes a name and a description of what will happen when that state is entered. Transitions include a description of the conditions that must be true in order for the transition to happen.

Fig 3-5. Simplified Flow Chart for Example 3.5 - "A Basic Task Program"



There are five states defined in this example Stateflow Chart, each of which is briefly described below.

TargetOn – When this state is entered, the first target will be turned on. The system will stay in this state until the condition “wait until hand is in target” is true, whereupon the system will switch to the `StayAtFirstTarget` state.

StayAtFirstTarget – Nothing happens when this state is entered. Once a specified period of time has elapsed, the system will switch to the `MoveTarget` state.

MoveTarget – When this state is entered, the target will be moved to the peripheral position (which is functionally identical to turning off the first target and simultaneously turning on the second target). The system will stay in this state until the condition “wait until hand is in target” is true, whereupon the system will switch to the `StayAtSecondTarget` state.

StayAtSecondTarget – Nothing happens when this state is entered. Once a specified period of time has elapsed, the system will switch to the `EndOfTrial` state.

EndOfTrial – When this state is entered, the target is turned off.

Other Task Program Controls to Add to Simplified Flow Stateflow Chart

Although the flow chart above defines the gross behavior of what can happen during a typical trial, there are other issues that typically need to be addressed to ensure proper Task Program control. Some of these include:

- an initializing state
- when to record data
- time limits on transitions to ensure that an infinitely long trial never occurs
- a Simulink event to indicate the end of trial (required by BKIN Dexterit-E)
- inter-trial timing

Each of these issues is addressed in the following figure. As before, no code is introduced yet, only the descriptions.

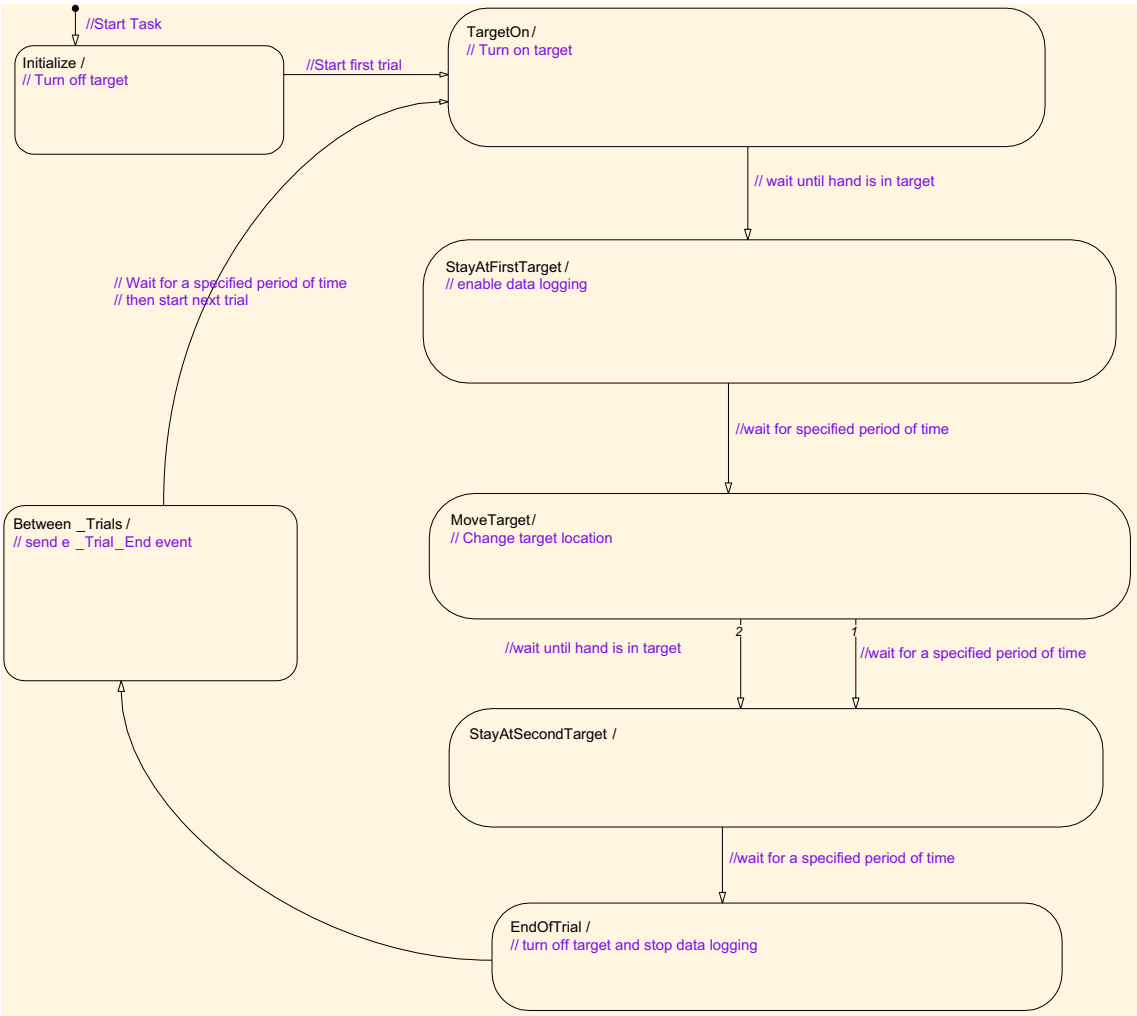
When a flowchart is first entered, there needs to be default starting location. It is good practise to initialize parameters at this point to ensure that the system is starting in the correct configuration.

The choice of when to record data will depend upon the purpose of the task. In this example, we have chosen to start recording data when the subject reaches the first (i.e. center) target. An advantage of this choice versus starting to record when the target is first presented is that if a subject decides to pause before ‘starting’ the trial, e.g. to talk to the user, then the amount of irrelevant data saved is reduced. A disadvantage would be for the situation in which the initial movement to the center target was of interest. Data recording is stopped in this example when the target lights turn off.

A time limit was added to each of the transitions that did not previously have a time limit to ensure that an infinitely long trial did not occur. This addition ensures stable behavior during a task for unexpected conditions (e.g. if there was a single target location that the subject could not reach to, but you still wanted to collect the other data without having to manually pause the task and force it to the next trial).

Inter-trial timing has been added, as well as an inter-trial state. The inter-trial state is necessary for BKIN Dexterit-E to function properly: it provides a time when the values of the Task Protocol parameters for one trial can be changed to those of the next trial.

Fig 3-6. Adding Task Program Controls to a Simplified Flow Chart for Example 3.5 - "A Basic Task Program"



There are now seven states in the Stateflow Chart. The new additions are described below:

Initialize – When the Task Program is first loaded, it needs to enter a default state, which in this case is the `Inititialize` state. The only action to be taken when this state is entered is to ensure that the target is initially off. There are no conditions for the transition leading to the `TargetOn` state, so the system immediately switches to the `TargetOn` state.

TargetOn – This state is the same as before.

StayAtFirstTarget – Data logging is now initiated when this state is entered (i.e. only data from this point on will be recorded for subsequent data analysis).

MoveTarget – There are now two possible conditions for exiting this state (if either is true, then the transition will occur). There is now a timing condition: once a specified period of time has elapsed, the system will switch to the StayAtSecondTarget state if the “wait until hand is in target” condition has not yet become true.

StayAtSecondTarget – Nothing new in this figure.

EndOfTrial – This state now transitions to a new `Between_Trials` state. There is no condition on this transition, so it occurs immediately.

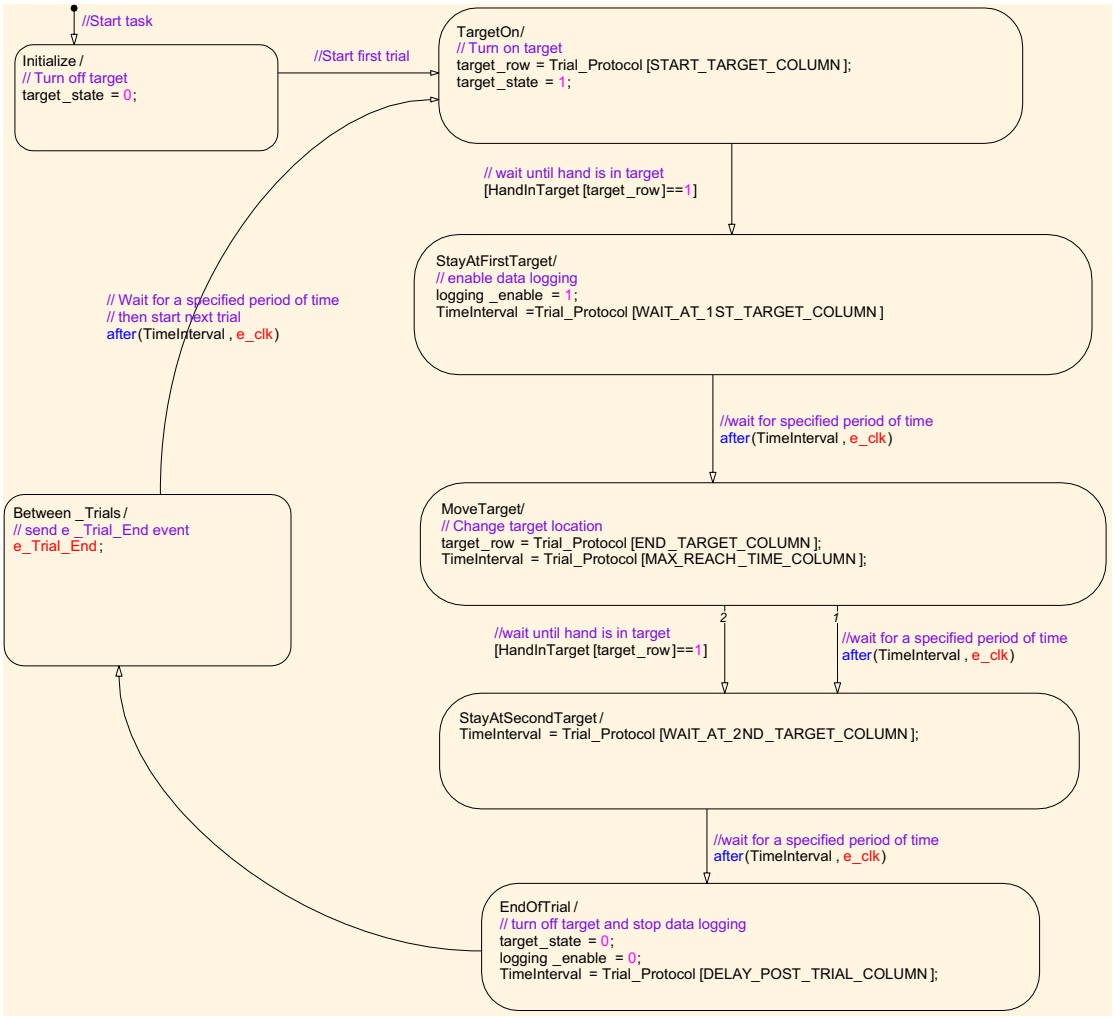
Between_Trials – This state is a place holder that occurs between trials. Its primary functions are (a) to send a Stateflow event out of the Stateflow chart to signal to the Task Program that the trial is over and (b) to provide a single time step delay, which allows the Task Program to update the `Trial_Protocol` for the next trial. Exiting of this state back to the `TargetOn` state for the next trial occurs after a specified time delay.

As a Task Program gets more complex, there are numerous other Task Program controls that a user may want to add at this pre-code stage: for example, event codes for post-data collection analysis (see Example 6.1 - "User-Defined Event Codes") or Pause button control (see Example 6.3 - "Pause Button Control (and Stateflow Sub-States and Stateflow Events)").

Final Stateflow Chart

In order to get to a usable Stateflow Chart, the necessary Stateflow code needs to be added to the states, transitions and descriptions that we have thus far provided. In the following figure, we have added the relevant code to implement the described actions and events. The code acts upon inputs to the Stateflow chart, and affects outputs from the Stateflow chart. In the code, these inputs and outputs appear as variables. For example, `Trial_Protocol` is an input and `target_row` is an output. There are also various local variables used, for example `TimeInterval` and constants, such as `START_TARGET_COLUMN`.

Fig 3-7. Final Stateflow Chart for Example 3.5 - "A Basic Task Program"



There are 7 states in this Stateflow Chart, each of which will be described briefly.

Initialize – When the Task Program is first loaded, it needs to enter a default state, which in this case is the "Inititialize" state, which simply ensures that the `target_state=0` (i.e. target is off)

TargetOn – The actions of this state result in a target being drawn by Simulink as soon as this state is entered. The choice of target is defined by the `target_row` and `target_state` outputs of this Stateflow chart (these outputs drive the `Show_Target` block in the Simulink diagram shown previously in **Figure 3-3**). `Trial_Protocol` is a vector input to the Stateflow chart and `START_TARGET_COLUMN` is a constant pointing to the appropriate index in the `Trial_Protocol` where the `target_row` for this target is stored. The `TargetOn` state is maintained until the required condition of "hand is inside the target" comes true, as defined by the [`HandInTarget[target_row]==1`] condition. `HandInTarget` is an input to the Stateflow chart which comes from a Simulink block that checks to see whether the hand is in any of the targets. If the hand is in the target associated with the present `target_row`, then the condition will be true and the Stateflow chart proceeds to the next state.

StayAtFirstTarget – This state is entered when the hand has reached the target. One action occurs upon entering this state: `logging_enable=1` is set. This setting tells BKIN Dexterit-E to start recording data (`logging_enable` is an output of the Stateflow chart, see **Figure 3-3**). The system

will then stay in this state until the exiting transition is true, which in this case is a time delay defined by the `after` function. The `after` function in Stateflow counts and waits for a prescribed number of Stateflow events. In this example, the prescribed number of events to count is `TimeInterval`, and the events being counted are `e_clk` events. For Task Programs, `e_clk` events are input Stateflow events that occur every 1 ms. (The `e_clk` event is sent by the `GUI_Control` block, and passed to the Stateflow chart through the event input located at the top of the Stateflow chart - see Section 9.8 - "GUI Control Input Events and Output Events" for more information).

MoveTarget – The actions of this state turn off the first target and turn on the second target. This action occurs simply by changing the value of `target_row` within the Stateflow chart. Because `target_row` is an output connected to the `Show_Target` block in Simulink, this change produces an effect in Simulink (see **Figure 3-3**). There are now two ways for the task to proceed: the task will wait until the hand is in this new target or until a specified period of time has elapsed before proceeding to the next state. The latter is defined by the `after(TimeInterval, e_clk)` condition.

StayAt2ndTarget – As with the `StayAtFirstTarget` state, this state simply adds a time delay before proceeding to the next state.

EndOfTrial – At the end of the trial, the `TimeInterval` variable is updated for the desired between-trial delay (used when exiting the subsequent state). Because there is no condition attached to the output transition from this state, the transition occurs automatically when the next Stateflow event occurs (`e_clk` events occur every 1 ms, so this transition will happen on the next `e_clk` event).

Between_Trials – This state is a place holder that occurs between trials. It has two primary functions. First it sends a Stateflow event (`e_TrialEnd`) out of the Stateflow chart to signal to the Simulink model that the trial is over. Second it provides a single time step delay, which allows the Simulink model to update the `Trial_Protocol` for the next trial. Transitioning from this state to the `TargetOn` state occurs after a time delay, determined by the `TimeInterval` updated in the `EndOfTrial` state.

Note:For the purposes of simplicity, this basic example does not allow for proper Pause button control by BKIN Dexterit-E. To learn how to add that functionality, please see Example 6.3 - "Pause Button Control (and Stateflow Sub-States and Stateflow Events)".

3.10 Setting Up `<your_task>_cfg.xml` for a Task Protocol

Each Task Program must have an associated `.xml` file (usually called `<your_task>_cfg.xml`). The purpose of this file is to define the column headings to be shown in BKIN Dexterit-E in the various parameter tables when defining a Task Protocol. Because each Task Program can have customized parameter tables, each Task Program must have its own `<your_task>_cfg.xml`.

`<your_task>_cfg.xml` for Example 3.5

The format of `<your_task>_cfg.xml` requires a single line for each column in each table. Within each line of the .xml file, there are several required tags, as described here and as shown in the example below. The headings and indices used in this file must match the Simulink/Stateflow code of the Task Program for proper functionality. The `<your_task>_cfg.xml` file for Example 3.5 - "A Basic Task Program" is shown here.

```
<?xml version="1.0"?>
<Config>

<TargetComponent index="1"type="float"name="X" desc="X Position (cm)"/>
<TargetComponent index="2"type="float"name="Y"desc="Y Position (cm)"/>
<TargetComponent index="3"type="float"name="VRad"desc="Target Radius (cm)"/>
```

```
<TargetComponent  index="4"type="float"name="LRad" desc="Logical Radius (cm)"/>
<TargetComponent  index="5"type="color"name="Color" desc="Target color"/>

<TPComponent      index="1"type="target"name="Start Target"        desc="First target"/>
<TPComponent      index="2"type="target"name="End Target"          desc="Second target"/>
<TPComponent      index="3"type="int"name="1st Target Wait"        desc="Hold (ms) at target 1"/>
<TPComponent      index="4" type="int"name="2nd Target Wait"       desc="Pause (ms) after target 2"/>
<TPComponent      index="5" type="int" name="Post-Trial Delay"     desc="Inter-trial pause (ms)"/>

</Config>
```

TargetComponent – indicates that this entry is for the Target Table

TPComponent – indicates that this entry is for the TP Table

index – indicates which column in the table is being defined

type – indicates what type of data will be entered in this column. Valid options are: “float” (floating point number), “int” (integer), “color” (only valid for data used for target colors), “target” (reference to the Target Table - only valid for a TP Table entry), “load” (reference to the Load Table - only valid for a TP Table entry)

name – name of the column to be shown in table (this name is also saved in the data file)

desc – description of the data in the column, available in the column table and also saved in the data file

For this example, only the `Target Table` and `TP Table` need to be defined (no loads are defined for this task, so no Load Table headings are defined). For each column in the Target Table and TP Table to be used, an entry in the .xml file is required. Note that the `index` values of the Target Table entries match those used in the Simulink code above in the `Show_Target` and `KINARM_HandInTarget` blocks, while the `index` values of the TP Table entries match those used in the Stateflow chart as defined in the Model Explorer.

Note: If you are building your Task Program in the final directory in which BKIN Dexterit-E will use it, there is the possibility of a conflict with .xml files. The build process that Simulink uses to compile a Task Program creates a .xml file that BKIN Dexterit-E can confuse with the <your_task>_cfg.xml file. To ensure that this conflict does not occur, we recommend building your Task Program in a different directory.

4 Compiling or 'Building' a Custom Task Program

Once a Task Program has been coded it needs to be compiled or “built” before it can be used with BKIN Dexterit-E. This chapter describes the basics of compiling a Task Program

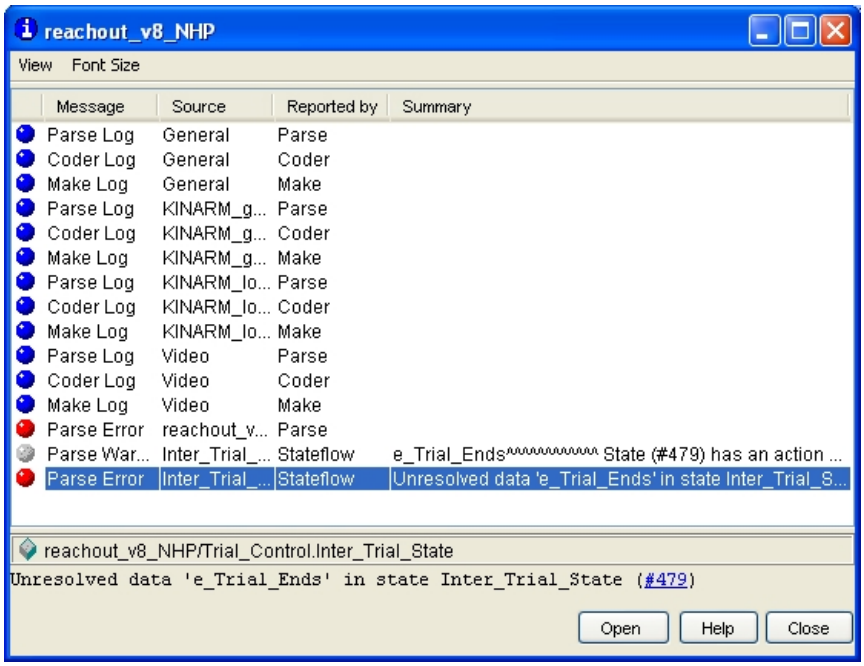
4.1 Building your New Task Program

In Mathworks’ terminology, “building” is what is done to compile a Task Program to a `<your_task>.dlm` file (which can then be used by BKIN Dexterit-E). In order to build a Task Program, `<your_task>.mdl` file must be open. In Simulink, choose: Tools→ Real Time Workshop→ Build Model (or press <Ctrl-B>) to start the build process. Any errors that occur will show up either in the Matlab workspace Command Window or in a new window. If the build process is successful, it produces a `<your_task>.dlm` file in the working directory, which you can then make accessible to BKIN Dexterit-E.

4.2 Common Errors During the Build Process

When an error occurs during the build process, the most common result is the creation a new window which lists the status and results of the build process, including the errors, an example of which is shown below. Depending on the error, however, feedback about the error may only appear in the Matlab Command Window. The following are some tips that can be useful in fixing common build errors.

Fig 4-1. Example of popup window created due to an error following the build process



General Tips for Build Errors

- Occasionally a long list of errors will be shown in the popup window following an attempt to build a new Task Program. Often, many of the errors will simply occur as a result of the first error, so a good strategy to try is to fix the first error and then rebuild the Task Program before attempting to fix the subsequent errors.

- Occasionally it will not be obvious from the popup window what the cause of the error is. In this case, updating the Task Program (rather than building it) can produce a different, more meaningful error message. To update your Task Program, right-click on the Simulink workspace of your Task Program and choose “Update Diagram”.
- Rarely, a Task Program will continue to fail to build properly, even after all apparent errors have been corrected. In this case, all of the temporary, intermediate files created by the build process should be deleted.
  a. Close Matlab.
  b. Delete the following files:
     - `<your_task>.xml`
     - `<your_task>ref.m`
     - `<your_task>pt.m`
     - `<your_task>bio.m`
     - `<your_task>_sfun.mexw32`
     - `*.lib`
     - folder named `<your_task>_xpc_rtw`
     - folder named `sfprj`
     - folder named `slprj`
  c. Restart Matlab and rebuild your Task Program.

Specific Build Error Messages

- `“Error using ==> xpc\private\xpcload Could not find target”`. If the preceding Build Error message appears in the popup window, then you need to uncheck an option within your Task Program. Open your Task Program, click on the Simulink menu option Simulation→ Configuration Parameters…, and in the left pane of the Configuration Parameters window select Real-Time Workshop→ xPC Target options. Uncheck Automatically download application after building. Rebuild your Task Program.
- `"sm_common_v2.c specified in custom source files string does not exist in any of the following directories: …"`. The `sm_common_v2.c` file was needed with TDK 1.1.x and earlier, but is not needed for later versions. To fix this problem, see Section 9.9 - "Updating Task Programs from Prior Versions of the TDK".
- `"…Cannot open include file: 'sm_common_v2.h': No such file or directory…"` The `sm_common_v2.h` file was needed with TDK 1.1.x and earlier, but it is not needed for later versions. To fix this problem, see Section 9.9 - "Updating Task Programs from Prior Versions of the TDK".
- `“Unresolved data ‘RAND_MAX in ...”`. The RAND_MAX value is defined in the stdlib.h file which must be included as part of the custom code library for the Stateflow chart in which the constant is called. See Section 6.18 - "Random Numbers in Stateflow" for more details.

5 Using and Testing a New Task Program

In order to use a custom Task Program, it must be made available to BKIN Dexterit-E. This chapter introduces the issues associated with making a Task Program available to BKIN Dexterit-E and initial testing of the Task Program.

5.1 Making your New Task Program Available to BKIN Dexterit-E

In order to make the new Task Program available to BKIN Dexterit-E, the following steps must be completed:

1. Copy the newly-created `<your_task>.dlm` file to a new sub-directory in the Task Programs directory on the computer running BKIN Dexterit-E (i.e. `"...\My Documents\Dexterit-E x.x Tasks\"`). Alternatively, if this is simply an updated `<your_task>.dlm`, then replace the previous version of `<your_task>.dlm`. with the updated version.
2. Copy over the associated `<your_task>_cfg.xml` file to the same directory. See Section 3.10 - "Setting Up <your_task>_cfg.xml for a Task Protocol" for more details.

   Note: If you are building your Task Program in the final directory in which BKIN Dexterit-E will use it, there is the possibility of a conflict in .xml files. The build process that Simulink uses to compile a Task Program creates a .xml file that BKIN Dexterit-E can confuse with the <your_task>_cfg.xml file. To ensure that this conflict does not occur, we recommend building your Task Program in a different directory.
3. Make sure that the version of Matlab used to create/build the new Task Program is the correct version. Task Programs can only be used with xPC Target boot disks made with same version of Matlab. If you are unsure about which version of xPC Target you are using, type `ver` in the Matlab Command Window. If you wish to start using a different version of xPC Target for you Task Programs, you must create a new xPC Target bootable disk. For details on how to create a bootable floppy disk, please consult the Dexterit-E User Guide.

5.2 Tips for Testing and Debugging a New Task Program

Once a Task Program has been successfully built, it must be tested using BKIN Dexterit-E. Task Programs cannot be tested within Simulink (i.e. you cannot simulate a running of a task within Simulink).

Second Mouse

If you are testing by yourself it can be difficult to calibrate the KINARM if you cannot reach the GUI computer from the KINARM. In order to make this easier you can acquire a second mouse (possibly with an USB extension cable) and plug it into the GUI computer. Placing a piece of opaque tape over the sensor of the second mouse will ensure that only its buttons work. With this set-up you can use the main GUI mouse to place the cursor, then carry the second mouse with you to the KINARM and use it to click through the calibration steps.

xPC Scopes

During the debugging phase of testing a Task Program, it is often useful to utilize xPC scopes. xPC scopes allow you to view Simulink signals numerically or graphically in real-time on the real-time computer. For example, if a load does not feel correct, you can view the commanded forces or torques being applied, as well as the signals being used to create those forces or torques. For more information, please see Example 6.6 - "xPC Scopes".

Disabling Loads

During the debugging phase of testing a Task Program, there exists the possibility of accidentally creating a dangerous loading condition. To avoid this problem, we recommend that you add xPC Scopes to the Task Program to view the loads, disable any motors on the system being controlled (e.g., with the system's Emergency Stop button), and then run the task with the motors disabled and view the commanded loads on the xPC Scopes. For more information, please see Example 6.6 - "xPC Scopes".

Separation of Stateflow and Simulink

The Stateflow code controls the logical flow of behavior during a task, whereas the Simulink code is more related to inputs and outputs that interact with a subject. It is useful to attempt to focus on these aspects of the code independently where possible. For example, if your task includes KINARM loads, by disabling the KINARM robots you can safely test and debug the behavioral logic in your Stateflow chart without having to worry about unsafe loading conditions.

Simplify the Task Program

If a new Simulink block has been created for your Task Program and it will be used multiple times in the Task Program, if possible it is best to have only one instance of the block in your task until it has been verified that the block works exactly as desired. This approach will not only simplify the debugging phase, but will also help ensure that incorrect, bug-filled copies of the block are not accidentally used.

Verifying Loading Conditions

If a Task Program produces loads, for example on a KINARM robot, the first test of correct load implementation in a Task Program is often based on perception of how the loads feel. Although this is a good first test, a few issues need to be considered when trying to interpret the perception of those loads.

- Is the testing being carried out at the point that the loads are being applied? For example, if the loads are defined for the KINARM robot at the 'fingertip', is the testing being done at that fingertip? If not, then a discrepancy will exist.
- Has testing being carried out in the absence of loads to notice the effects of robot inertia? The KINARM robot's inertia ellipse is aligned with that of the arm (i.e. the shoulder joint is 'heavier' than the elbow 'joint'). If the robot is being held at the end-point, the effects of this inertia ellipse can be felt in two ways. The first is the direct effect of the inertia on force (higher forces will be required to move the KINARM robot along the major axis of inertia versus the second). The second is an indirect effect of the inertia - unless carefully controlled, movements along the major axis of inertia will be slower, and if the load is a velocity-dependent load, the result will be less of a load.
- For the KINARM robots, the loads commanded by a Task Program are applied to the robot, which in turn applies the loads to the subject. Under static conditions these loads are identical, but under dynamic conditions, the mass and inertia of the robot can change the loads slightly due to Newton's equations of motion (i.e. F = ma). There are two methods of examining this issue. The first is to perform the same experiment in the absence of loading conditions to 'feel' the effects of moving the KINARM robot in different directions and then compare that to the loaded conditions. The second approach is to determine the effects analytically, which requires one to implement the equations of motion and calculate the effects of the KINARM robot.

WARNING!

When testing a new or newly-edited Task Program, appropriate safety precautions must be taken with any system that can produce loads, such as a KINARM. This includes having the system's Emergency Stop button readily accessible and/or keeping a firm grip on the robot being controlled. Mistakes in coding can result in potentially dangerous behavior of the system, resulting in either damage to the system and/or injury to a subject and/or operator.

5.3 Errors When Trying to Run a New Task Program in BKIN Dexterit-E

If your Task Program compiles successfully and you successfully load it in BKIN Dexterit-E and nothing happens, then follow these steps to assist in identifying the problem.

1. Switch the monitor/keyboard to the xPC Target machine (2nd computer that runs the Task Programs). Although it is possible to have this computer set up with its own monitor/keyboard, typically it shares a monitor/keyboard with the BKIN Dexterit-E Window's computer via a KVM switch. Switch to the xPC Target machine by either pressing a button on the KVM switch or by pressing the <Scroll Lock> key twice (taps must be in quick succession, just like a double-click on the mouse).
2. Does the error "CPU Overload" occur? If so, then see notes below.

5.4 Error - "CPU Overload"

If you receive this error on the xPC Target computer, then your Task Program has taken too long to execute during a single clock cycle (500 microseconds). There are many multiple causes/fixes for this error (please see Mathworks for causes other than those listed here):

- Some very CPU intensive operations have been added (and will need to be rewritten to be more efficient). This is not a common cause, but if using an embedded M-file, be sure to take advantage of Matlab's vector notation for optimal speed, otherwise this can be an issue.
- Too fast a sampling rate of the Task Program - the default execution rate for Task Programs is typically 2 kHz (From within your Task Program in Simulink, choose Simulation → Configuration Parameters. In the "Configuration Parameters" window choose Solver on the left-hand pane and look at "Fixed-step size (fundamental sample time)". The default value is 0.0005. If you have increased the rate beyond 2 kHz (e.g. to 4 or 8 kHz), then this could be the source of the problem.
- Too many AI channels or too slow acquisition time - data acquisition of analog input channels can take a significant time. The amount of time taken by analog input data acquisition is equal to the number of channels of analog input multiplied by the conversion rate (sampling time) for each channel. For example, if the per-channel conversion time is 12.8 microseconds and 32 channels are being recorded, the total acquisition time is 12.8 microseconds * 32 = 410 microseconds, which for a 2 kHz Task Program execution rate, leaves only 90 microseconds for everything else in the Task Program (as well as xPC overhead). Try reducing the number of analog input channels to determine if this is the source of your problem. If it is the source of your problem, but you cannot reduce the number of channels of analog input without compromising your needs, then the mean conversion time will need to be reduced.

  NI PCI-6071E (not PXI-607IE). To change the mean conversion time for this card, edit `adnipcie.c` in the TDK directory. Search for:

  ```
  …
  case 8:
  strcpy(devName,"NI PCI-6071E");
  devId=0x1350;
  resFloat=4096.0;
  convDelay=0x0100;
  break;
  …
  ```

  and change the convDelay to a smaller number. The parameter `convDelay` is the delay in NI `Time_clock` periods (20 MHz time clock). In the above example, 0x0100 is the hexadecimal equivalent of 256, which converts to a per-channel sampling time of 12.8 microseconds (i.e. period of 20 MHz clock is 0.05 microseconds, so 256 periods of this clock produces a total time of 12.8 microseconds). Although

the PCI-6071E is rated to 12 bits accuracy for source impedance < 1kOhm using 5 microseconds conversion time, greater than 1 kOhm source impedance requires greater than 5 microseconds for 12 bits accuracy (and likewise less than 1 kOhm source impedance will likely require less than 5 microseconds). The choice of an acceptable conversion delay will depend on the desired accuracy and the source impedance of the analog signals being recorded.

6 Examples of Task Program Code

This chapter provides numerous examples of code to highlight features common to many Task Programs. Most of these examples build upon the basic example shown in Section 3.5 - "A Basic Task Program".

6.1 User-Defined Event Codes

User-defined event codes provide a method of time-stamping when something occurred and what that something was, which can be useful for subsequent data analysis. Examples of typical events that are time-stamped in this manner include target on/off events and user-defined errors (e.g. the subject moved too slowly), however, they can be anything that the user wishes. User-defined event codes are something that are end-user specific; they have no meaning to BKIN Dexterit-E. User-defined events are displayed in the BKIN Dexterit-E GUI as they occur during run-time.

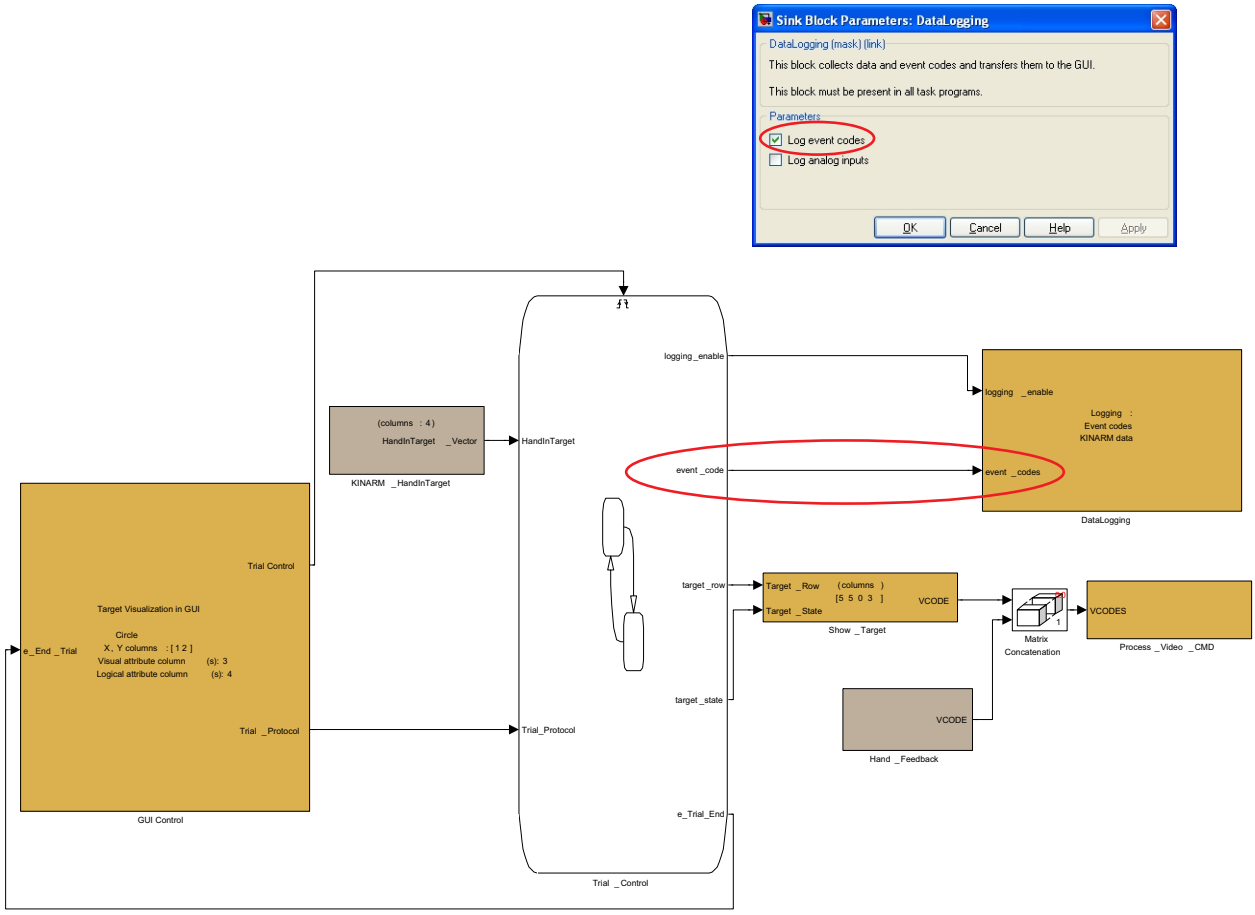
To save a user-defined event, a user-defined event code must be passed into the DataLogging block in the Simulink code (see example below). User-defined event codes are logged only when the `event_codes` input to the `DataLogging` block changes. User-defined event codes are 16-bit integers (however, if using an external DAQ system such as Plexon, then only the lower 15 bits are saved). To save a textual meaning to an event code (as opposed to just the 16-bit integer), a name and description for the event code must be defined in the `<your_task>_cfg.xml` file (see example below).

Simulink Code for Example 6.1

This Task Program is the same as the example of the basic task in Chapter 3 - "From Concept to Code: Converting an Idea into a Task Program", but with a new output added to the Stateflow chart called `event_code`. This new output is connected to the `event_codes` input of the `DataLogging` block, which has been made available by checking the `Log event codes` check box in the `DataLogging` block's dialog. It allows user-defined events to be recorded during a task.

Note:As with all other data logging, event codes are only recorded when the logging_enable input of the DataLogging block is set equal to 1.

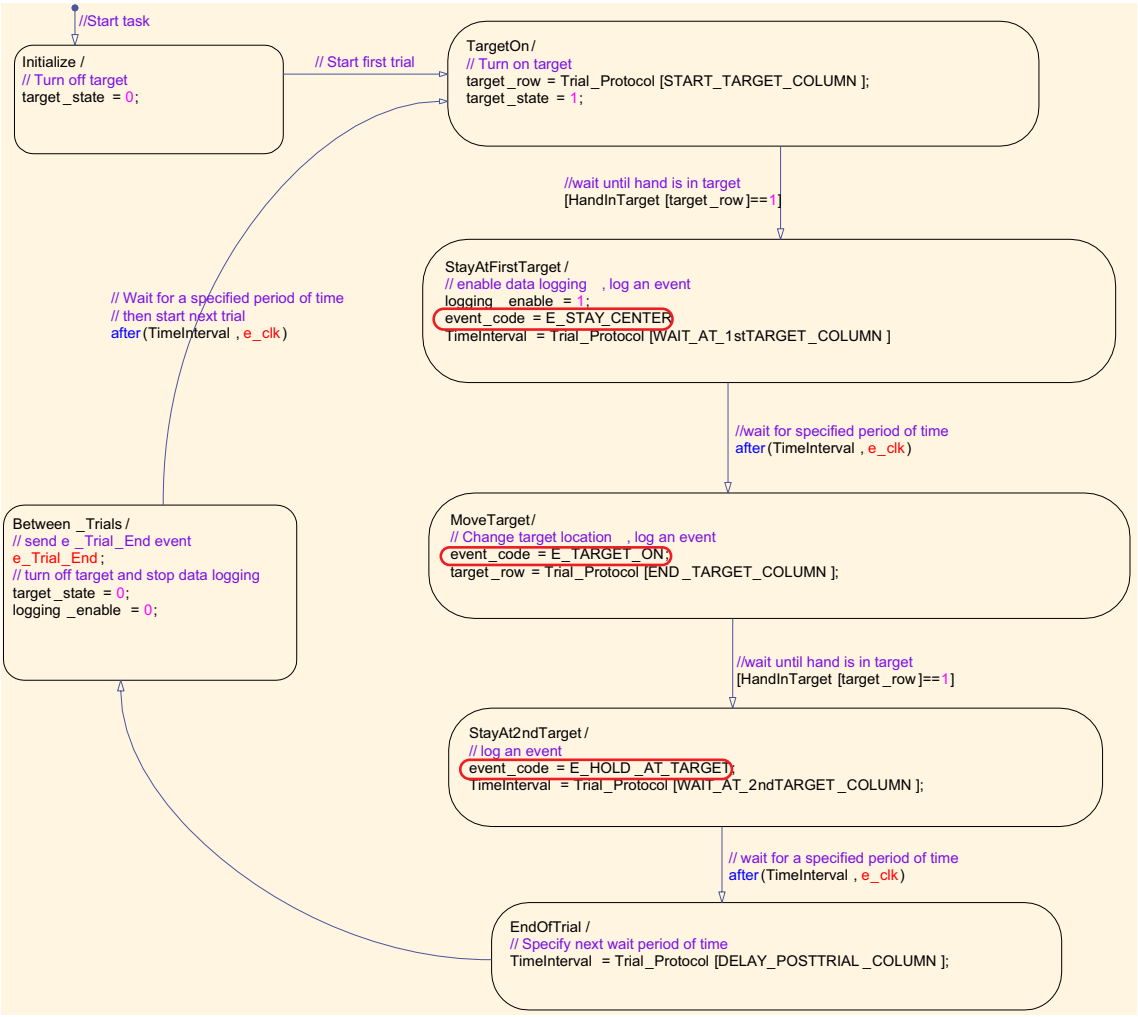
Fig 6-1. Simulink Code for Example 6.1 - "User-Defined Event Codes"



Stateflow Chart for Example 6.1

In the Stateflow Chart shown next, the output `event_code` is assigned various values when different states are entered. The values that `event_code` is set to in this example (e.g. `STAY_CENTER`) are constants, defined in the Model Explorer.

Fig 6-2. Stateflow Chart for Example 6.1 - "User-Defined Event Codes"

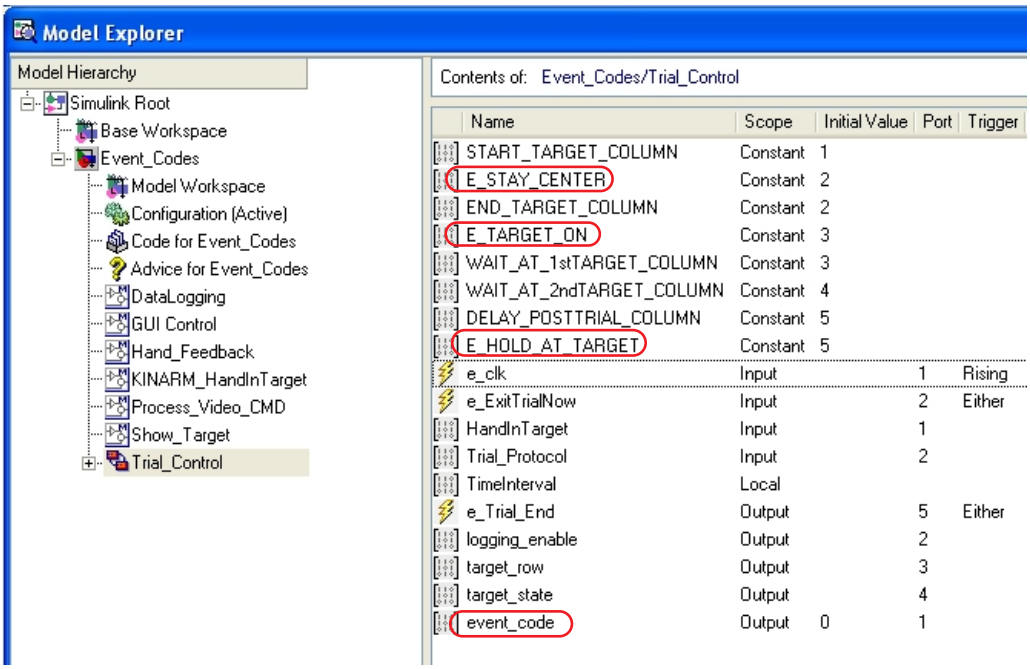


Events are only registered and saved when the `event_codes` input to the DataLogging block changes. It is therefore imperative that the `event_code` be forced to change at least once per trial. In this Stateflow Chart example, there is only one branch to the flow chart, and so the `event_code` will cycle through different values and will always be set appropriately. However, for a StateFlow Chart with multiple branches, setting `event_code=0` as part of the `Between_Trials` state might be required. See Example 6.3 - "Pause Button Control (and Stateflow Sub-States and Stateflow Events)" for an example of this type.

Model Explorer for Example 6.1

As compared to the basic Task Program described above, several new constants have been defined (e.g. `E_TARGET_ON`) and a new output (i.e. `event_code`) has been created. Note that the new `event_code` output has been given an initial value of 0. This value was chosen to ensure that when the first desired user-defined event occurs, that the `event_code` output will change its value (the Simulink code only registers a user-defined event when the `event_code` changes). Alternatively, a line could have been added to the `Initialize` state in the Stateflow chart such as `event_code=0` to achieve the same effect.

Fig 6-3. Model Explorer for Example 6.1 - "User-Defined Event Codes"



`<your_task>_cfg.xml` for Example 6.1

To save a meaning for user-defined event codes, the end-user must edit the `<your_task>_cfg.xml` file. This `.xml` file is used by BKIN Dexterit-E to interpret user-defined event codes. Each user-defined event code should have an associated line in the `.xml` file as per the following:

```
…
<Event code="2"      name="STAY_CENTER"                desc="Subject has reached 1st tar-
get"/>
<Event code="3"      name="TARGET_ON"            desc="2nd target light on"/>
<Event code="5"      name="HOLD_AT_TARGET"             desc="Subject has reached 2nd tar-
get"/>
…
```

Even – indicates that this line is defining a user-defined event

code – is value of the user-defined event code used in Simulink (i.e. when the Task Program logs an `event_code` of this value, then the name and description associated with that code as defined here is what gets saved in the data file)

name – as described above for Section 3.5 - "A Basic Task Program"

desc – as described above for Section 3.5 - "A Basic Task Program"

In this example, the rest of the `<your_task>_cfg.xml` file looks the same as the Section 3.5 - "A Basic Task Program", with these event-related lines added in.

6.2 Task Control Buttons

For some tasks it is useful to have the operator able to provide input in order to mark events or advance trials. It is possible for a task to define buttons that are displayed in BKIN Dexterit-E. The buttons are defined

within the <your_task>_cfg.xml file. Button presses during task execution are recorded automatically in the c3d files as events. Here is an example of defining buttons:

```
<Eventcode="0"      name="BETWEEN_TRIALS"   desc="Between trials"/>
<Eventcode="2"      name="STAY_CENTRE"      desc="Subject waits at centre starting now"/>
<Eventcode="3"      name="TARGET_ON"        desc="Target light ON"/>
<Eventcode="5"      name="HOLD_AT_TARGET"   desc="Subject starts holding at the target"/>

...

<Button index="10" name="Start trial" desc="Starts next trial." enable_events=”2”
disable_events=”3”/>
<Button index="11" name="End trial" desc="Ends current trial" enable_events=”3”
disable_events=”5”/>

</Config>
```

The variables in the button XML tag have the following meanings:

index – This is an identifier for the button. Buttons are placed in the GUI in order of ascending index. Within the saved c3d files this number is stored as an event in the event codes section when the button is pushed. Be sure to use a number that is different from any event code you plan to use. This may be any value from 2 to 254.

name – The text that will appear on the button in the GUI

desc – The text that will appear beside the button in the GUI

enable_events – A comma separated list of defined events that will cause the button to enable. If there are no enable_events defined then the button is enabled by default.

disable_events – A comma separated list of defined events that will cause the button to disable.

In order to make use of your defined button pushes you will need to use a “from” tag in your simulink code. In the From tag select “task_control_button”. When the operator clicks a button in the GUI the from tag in simulink will output the index value of the button for one cycle. The rest of the time the from tag will output a zero.

Here is an example of a simple button which when pushed will produce a Simulink event that can drive a Stateflow chart (e.g. to advance to the next trial within the Stateflow char).

```
<Button index=”10” name=”Continue” desc=”Click to move to the next trial”/>
```

Fig 6-4. Example task control button in the GUI.

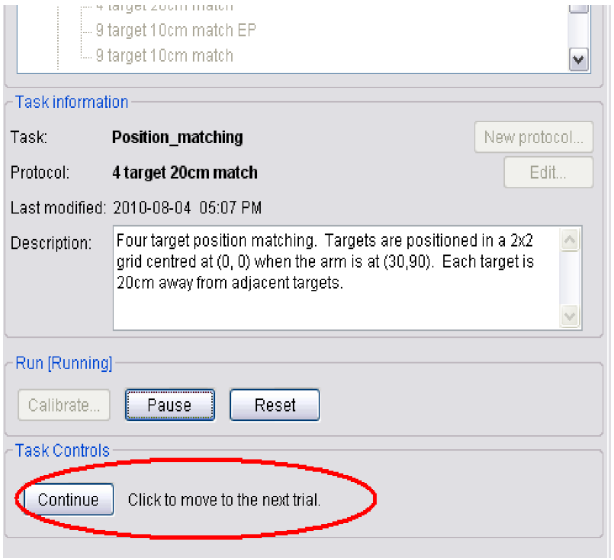
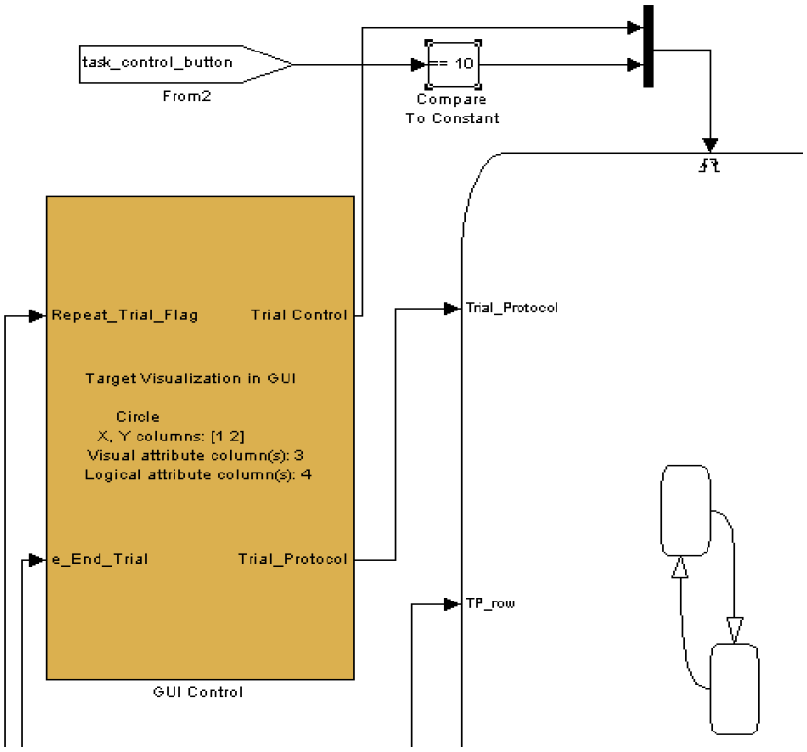


Fig 6-5. Using a task control button in Simulink



6.3 Pause Button Control (and Stateflow Sub-States and Stateflow Events)

The purpose of this example is to demonstrate the Stateflow chart requirements in order for the Pause button in BKIN Dexterit-E to work properly. The action caused by clicking Pause during a task depends on

the choice made by the user in BKIN Dexterit-E (see Dexterit-E User Guide for more information). There are five possible Pause actions defined for BKIN Dexterit-E:

- Pause immediately, continue with trial when unpaused
- Pause immediately, restart trial when unpaused
- Pause immediately, go to next trial when unpaused, do not repeat paused trial
- Pause immediately, go to next trial when unpaused, repeat paused trial at end of block
- Pause at end of trial, continue with next trial when unpaused

For the two Pause options that Pause the trial immediately and then do not complete the trial, the Stateflow chart diagram below demonstrates what code is needed in the Task Program for this functionality. If the elements described below are not present, then the Pause button will not behave as expected; instead the trial will pause immediately and then continue with the same trial when unpaused.

Note:A "Task Paused" event will be stored in the output data file to record the start of the pause. All data logging halts when a task pauses.

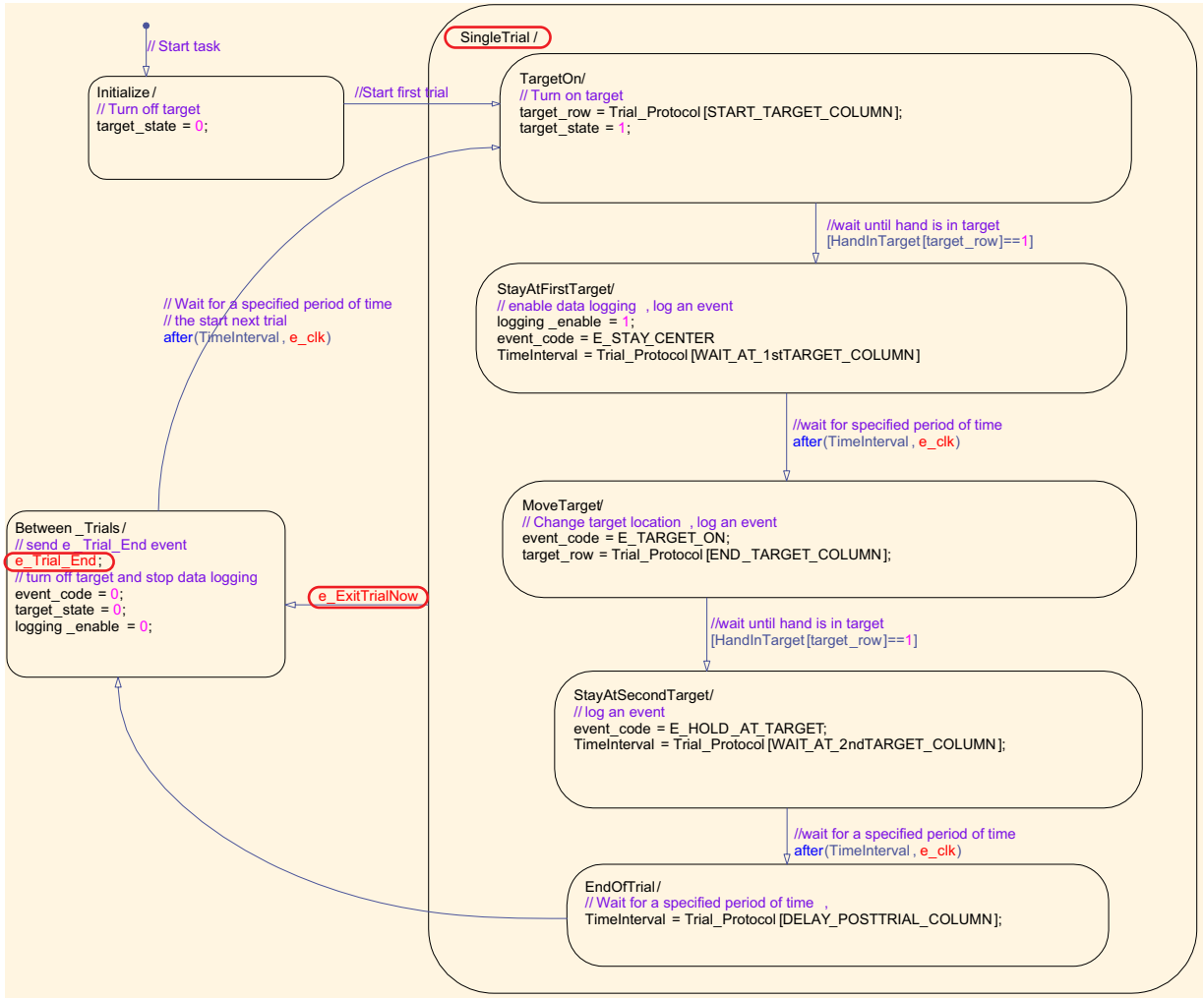
Simulink Code for Example 6.3

The Simulink code for this example is not shown here. It is the same as that shown in Section 3.5 - "A Basic Task Program".

Stateflow Chart for Example 6.3

For Pause button control to work, at least two Stateflow input events must be defined for the Stateflow chart. The `Trial_Control` output of the `GUI Control` block is composed of these two Stateflow events: a 1 kHz clock and an event that can occur when the Pause button within BKIN Dexterit-E's GUI is pressed. In this example, the second input Stateflow event is named `e_ExitTrialNow`. This `e_ExitTrialNow` event only occurs when the chosen pause-action requires an immediate exit from the trial. For example, if the Pause button control in BKIN Dexterit-E is set to Pause immediately, restart trial when Unpaused, then `e_ExitTrialNow` will occur. As can be seen in the Stateflow Chart below, the occurrence of an `e_ExitTrialNow` event causes a transition to the `Between_Trials` state from any of the sub-states within the `SingleTrial` state. The key features to implement this behavior are required: (a) a new parent-state (e.g. `SingleTrial` state) must be created which encompasses all of the other states except for the `Between_Trials` state (b) a transition must be created from this new state to the `Between_Trials` state which occurs upon the second input Stateflow event (i.e. `e_ExitTrialNow`).

Fig 6-6. Stateflow Chart for Example 6.3 - "Pause Button Control (and Stateflow Sub-States and Stateflow Events)"



The significance of exiting to the `Between_Trials` state is that it contains the output Stateflow event `e_Trial_End`, which is the Stateflow event sent to the `GUI Control` block to indicate the end of trial. Also note that in many Task Programs, various actions might need to be included as part of the new exiting transition to ensure that the system exits smoothly and appropriately. For example, if large loads are being used, the implementation in this example will cause the loads to turn off immediately, which could be uncomfortable or even dangerous. Extra states and transitions to scale the load down over some given time period upon the e_ExitTrialNow event can avoid such an issue (not shown here).

Note: The Between_Trials state now has an event_code=0 command. This command ensures that during the next trial, events will be registered properly even if the e_ExitTrialNow event occurs. See Example 6.1 - "User-Defined Event Codes" for more information.

6.4 Analog Inputs

The purpose of this example is to show what basic changes are necessary to have a Task Program record and save analog inputs. The purpose of such a setup is to be able to record analog information (e.g.

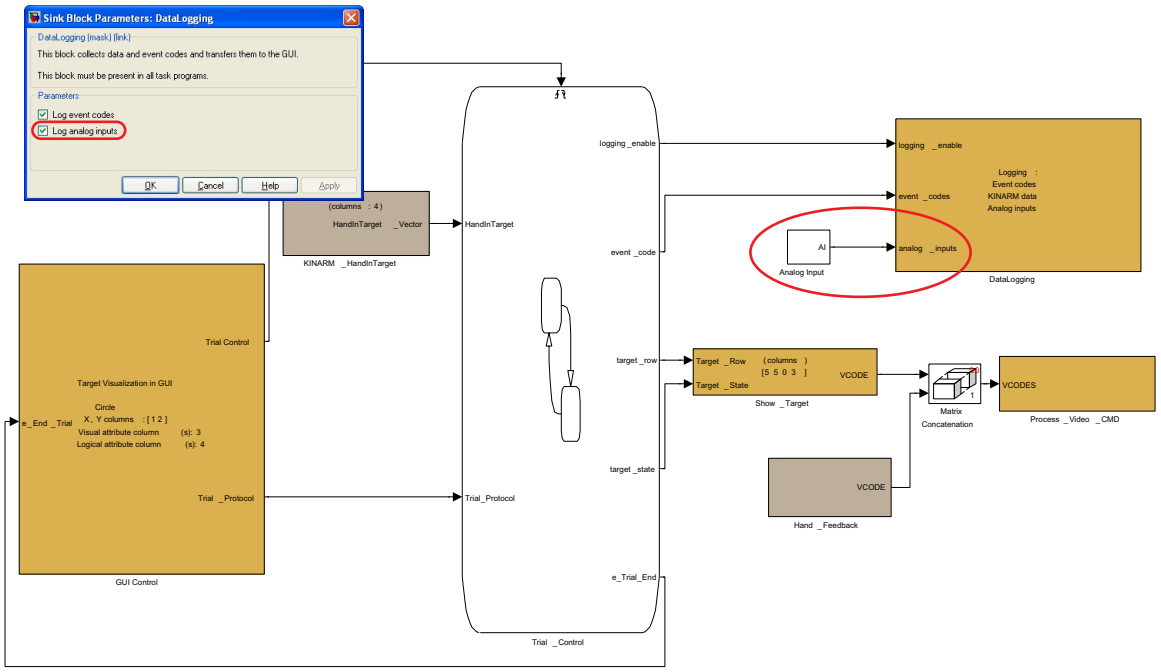
electromyogram, torque outputs from the motor's servo amps) synchronously with other information such as kinematic information from the KINARM robots.

Note:Recording of analog inputs requires an analog input card, such as that suggested with the standard BKIN Dexterit-E requirements. For more information, refer to the Dexterit-E User Guide.

Simulink Code for Example 6.4

In this example, the “Analog Inputs” option in the `DataLogging` block has been checked, such that a new inport has appeared in the `DataLogging` block. A custom, user-defined block corresponding to the hardware used for data acquisition (called Analog Input in this example) is added to the Simulink diagram, and then configured and wired into the `DataLogging` block. As with all other data logging, the `analog_data` are only recorded when the `logging_enable` input of the `DataLogging` block is set equal to 1.

Fig 6-7. Simulink Code for Example 6.4 - "Analog Inputs"



Note:This portion of the Simulink diagram must be present to record the actual torque values of the motors if desired. By default, only the commanded torques are recorded as part of the “KINARM data”.

6.5 Loads on the KINARM Robot

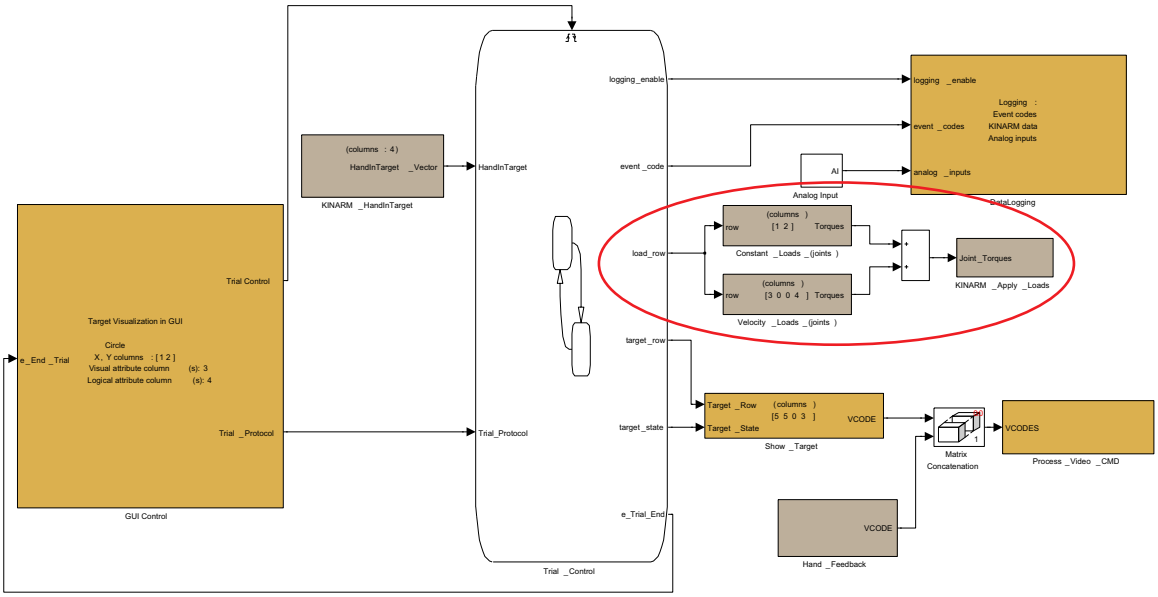
The purpose of this example is to show the Simulink blocks necessary to apply loads to a KINARM robot. In this example, the sum of two different classes of loads are to be applied to a KINARM Exoskeleton robot: constant loads and velocity dependent (e.g. viscous) loads.

The Simulink blocks used to create and apply loads in this example are available in the `KINARM loads` library of the BKIN Task Development Kit. For KINARM End-Point robots, analogous blocks are available in the `KINARM EP loads` library.

Simulink Code for Example 6.5

Implementing a load on the KINARM robot requires at least 2 blocks: a load block that creates a torque output (e.g. `Constant_Loads_(joints)`) and the `KINARM_Apply_Loads` block, which applies the desired torques to the KINARM robot through a motion control card. Multiple simultaneous loads can be applied as shown in the example below, by copying additional load blocks from the BKIN Task Development Kit library and summing the output torques prior to inputting the torque to the `KINARM_Apply_Loads` block. The actual parameters to be used for each load are gathered from the `Load_Table`, as referenced by the column indices chosen by the block (in this example, `Constant_Loads_(joints)` will reference columns [1 2]) and also by the “row” input to the block from the Stateflow chart (i.e. the Stateflow chart has a new output: `load_row`). For more details on load and column definitions, click the help button of the “load” block of interest within Simulink.

Fig 6-8. Simulink Code for Example 6.5 - "Loads on the KINARM Robot"



Note:There can be only one copy of the KINARM_Apply_Loads block in any Task Program.

Stateflow Chart for Example 6.5

The Stateflow Chart for this example is not shown here. The main feature is a new Stateflow output `load_row`, which must be set as desired from within Stateflow (e.g. `load_row = Trial_Protocol[LOAD_COL]`, which assumes that the load choice is defined in the `Trial_Protocol`)

<your_task>_cfg.xml for Example 6.5

Although the Simulink code completely defines how coefficients in the Load Table will get passed to the various Load blocks in the Task Program, in order for the Load Table to appear properly for editing in BKIN

Dexterit-E's Windows-based GUI, the `<your_task>_cfg.xml` file needs to be edited. For each column in the Load Table that is to be used, a line needs to be included in the `.xml` file. Below are the four lines that would be included for this “Loads” example (only four columns from the Load Table are used by the Simulink code: columns 1 and 2 for the `Constant_Loads_(joint)` block and columns 3 and 4 for the `Velocity_Loads_(joint)` block).

```
…
<LoadComponent  index="1"type="float"name="Sh bias(Nm)"     desc="Shoulder torque"/>
<LoadComponent  index="2"type="float"name="Elb bias(Nm)"desc="Elbow torque"/>
<LoadComponent  index="3"type="float" name="Sh visc(Nm/(rad/s)" desc="Shoulder viscosity"/>
<LoadComponent  index="4"type="float" name="Elb visc(Nm/(rad/s))"      desc="Elbow viscosity"/>
…
```

LoadComponent – indicates that this line is defining a column in the Load Table

index – as described above for Example 3.5 - "A Basic Task Program"

type – as described above for Example 3.5 - "A Basic Task Program"

name – as described above for Example 3.5 - "A Basic Task Program"

desc – as described above for Example 3.5 - "A Basic Task Program"

In this example, the `<your_task>_cfg.xml` file looks the same as Example 6.1 - "User-Defined Event Codes", but with the above four lines added to it.

Note:Loads are applied to the KINARM robot, not to the subject, whether the loads are hand-based or joint-based. Under static conditions the load applied to the KINARM robot will equal the load applied to the subject, but under dynamic conditions these will not be equal because of Newton’s laws and the equations of motion.

6.6 xPC Scopes

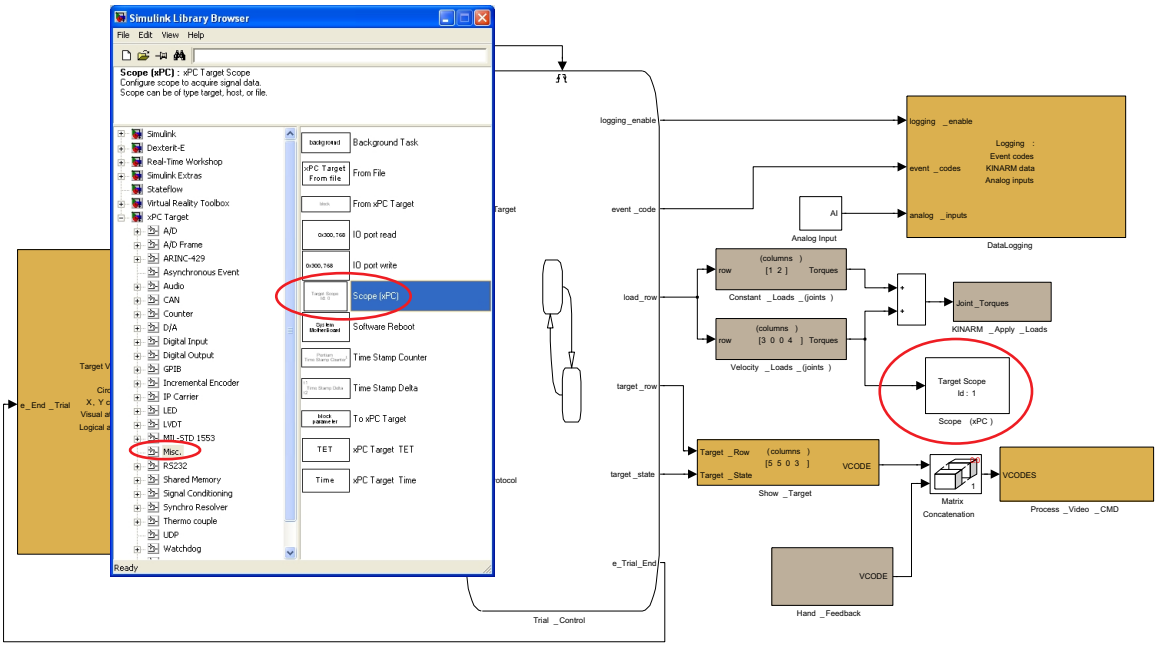
The purpose of this example is to show the use of an xPC Scope in a Task Program. xPC Scopes allow the user to view signals during Task Program execution on the real-time computer. This feature can be useful when debugging a Task Program. In a typical setup, where the real-time computer and BKIN Dexterit-E GUI computer share a keyboard, video and mouse, the KVM switch will need to be used to switch the monitor to view the real-time computer (e.g. double-click <Scroll Lock> to switch which computer is viewed on the monitor).

Note:The Scope (xPC) block should not be confused with the standard Simulink Scope block.

Simulink Code for Example 6.5

To use an xPC Scope, the `Scope (xPC)` block can be dragged from the Simulink Library Browser; the block is found at `xPC Target→ Misc→ Scope (xPC)`). Once it has been dragged into the Task Program, a signal can be connected to the xPC Scope. To view multiple signals on the same scope, the signals must be muxed together first. To view different signals on different scopes, drag multiple copies of the `Scope (xPC)` block into the Task Program. Double-clicking on the xPC Scope will bring up a dialog (not shown) with numerous options, such as whether to show numerical or graphical data, number of samples to show, triggering options etc.

Fig 6-9. Simulink Code for Example 6.6 - "xPC Scopes"



6.7 Multiple Targets and Multiple Target States

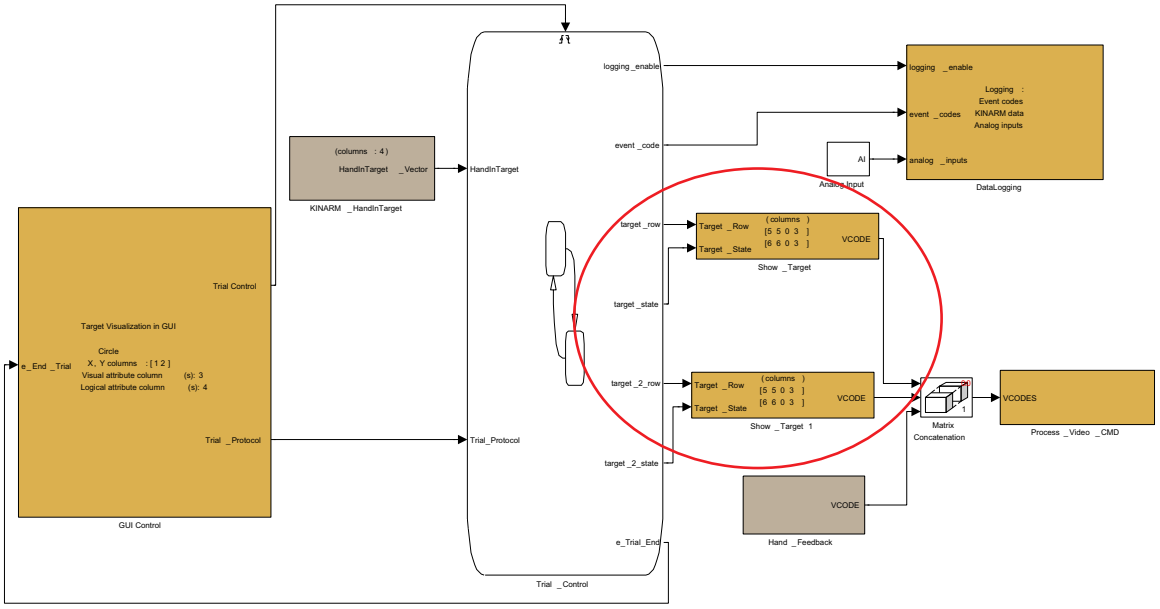
The purpose of this example is to show how multiple targets can be displayed simultaneously and to show how multiple target states can be used. The purpose of having multiple “on” states for a target is to allow a single target to change from one color and/or size to another without having to completely redefine the target. For example, if you wish to have a target change from green to red, then a target can be defined with two states such that the two states share the same location, size and target type, and the only difference between them is the color.

Simulink Code for Example 6.7

Multiple states for a target are defined by double-clicking the `Show_Target` block within Simulink, and choosing 2 or more states. For each desired state, the columns in the `Target_Table` that reference size and color need to be chosen. Having the target switch back and forth between these states then requires the `Target_State` input to the `Show_Target` block to be the desired state. For more information on defining target states, click Help in the `Show_Target` block from within Simulink.

Showing multiple targets simultaneously requires two things: the `Matrix Concatenation` block (which concatenates the `VCODE` outputs) and a separate `Show_Target` block for each target. In this particular example, 2 targets are shown.

Fig 6-10. Simulink Code for Example 6.7 - "Multiple Targets and Multiple Target States"



Stateflow Chart for Example 6.7

The Stateflow Chart for this example is not shown here. The main features are two new outputs `target_2_row` and `target_2_state` which must be set as desired from within stateflow in a similar manner to target_row and target_state. Also note that `target_state` and `target_state_2` now have three valid values: 0 (target off), 1 (state 1) and 2 (state 2).

<your_task>_cfg.xml for Example 6.7

```
…
<TargetComponent  index="1”type="float"name="X" desc="X Position (cm)"/>
<TargetComponent  index="2"type="float"name="Y" desc="Y Position (cm)"/>
<TargetComponent  index="3"type="float" name="VRad"desc="Target Radius (cm)"/>
<TargetComponent  index="4"type="float" name="LRad"desc="Logical Radius(cm)"/>
<TargetComponent  index="5"type="color"name="Color" desc="Target color"/>
<TargetComponent  index="6"type="color"name="Color2" desc="Color after target reached"/>
…
```

In this example, another column has been defined in the Target Table. State 2 of the `Show_Target` blocks reference column 6 of the Target Table for the color of the targets (see Simulink code above) and so a new line needs to be added to the `<yourtask_cfg>.xml` for `index=”6”`.

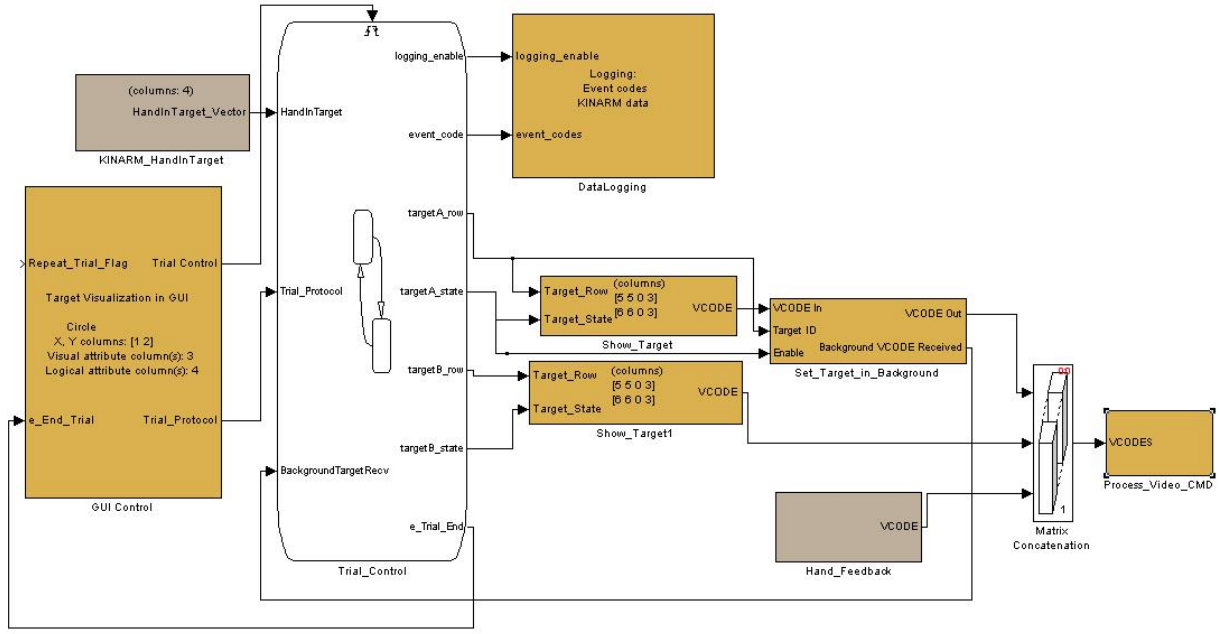
You can either set the transparency (or alpha) for a target within a simulink show target block, or have it passed into the show target block as a parameter. Within the show target block you can also specify if a target should show up on the operator display (i.e. in Dexterit-E), the subject display, or both.

6.8 Permanent targets

If you need to display more than a few targets at the same time and certainly if you need to display more than 15 targets at a time then you should consider using preeminent or background targets. These are targets which are defined just like any other target, but they are pushed through a “set target in background” block.

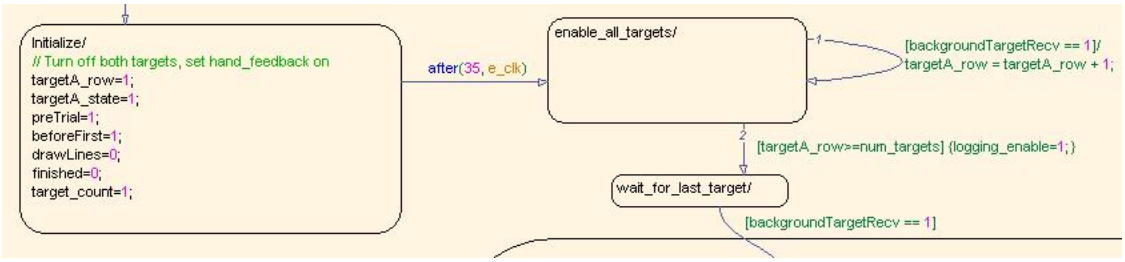
Simulink Code for Example 6.8

This simulink shows how to use the “set target in background” block. It requires the VCODE that is created in a normal “show target” block, an ID to specify which background target to deal with, and a state to turn it on or off.



There is also a signal fed back from the “set target in background” block into the state flow diagram. This is a pulse which is sent so that you know the target update has been received and you can continue. This signal goes to 1 for 1 cycle when a background target command has been received. This is absolutely required, otherwise any changes you make to the target are not likely to actually take effect. Generally this should take about 1 v-sync, or 17ms for a 60 frame per second display.

Here is an example of the stateflow that you would use to turn on a set of targets:



6.9 Targets with Text

You can display targets with text. To do this you need to define 3 columns for the target table in your <task name>_cfg.xml file. One column is of type 'label', another is of type color for the text color, and a float column for the size of the text in cm. This data needs to be passed to a 'Show Target with Label' simulink block within your model. Within Dexterit-E you will be able to define the text. The text will always be shown centered over the target you have specified and at the size you have specified. The text can only be a maximum of 70 characters long.

6.10 Images as Targets

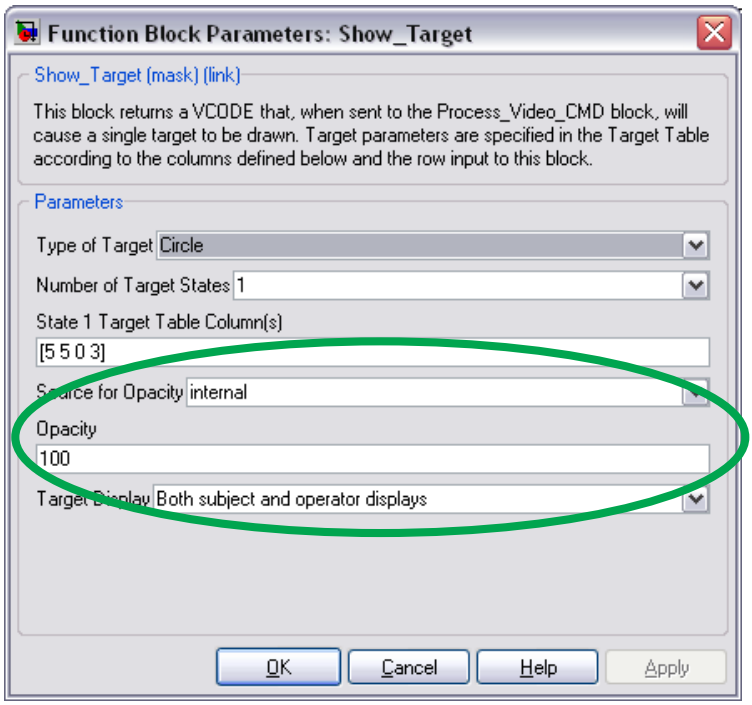
It is possible to show any jpg, png, or bmp as a target. This is mainly handled through Dexterit-E. When you define your target table in <task name>_cfg.xml any column of type "colour" or "color" can be used within the protocol editor in Dexterit-E to display images or colors. Only images which are stored in the task's directory will be available. The image will be displayed mapped onto the shape you have specified in the target table. If you need to manipulate VCODES yourself then be aware that negative values in color are interpreted as image indexes. Images are indexed in alphabetical order within the task directory. This also means that you cannot include an alpha value in your color directly, there is a different place in the VCODE to add this.

6.11 Selective Target Display Options

It is normal that most targets are shown both on the subject display and within Dexterit-E. However, it is possible to display targets only to the subject, or only to the operator in Dexterit-E. These are options which are set within the block mask for a "Show_Target" or "Show_Target_With_Label" block.

It is also possible to apply a transparency setting to targets within the "Show_Target" or "Show_Target_With_Label" block. This allows one target to be seen underneath another.

Fig 6-11. Show_Target Options



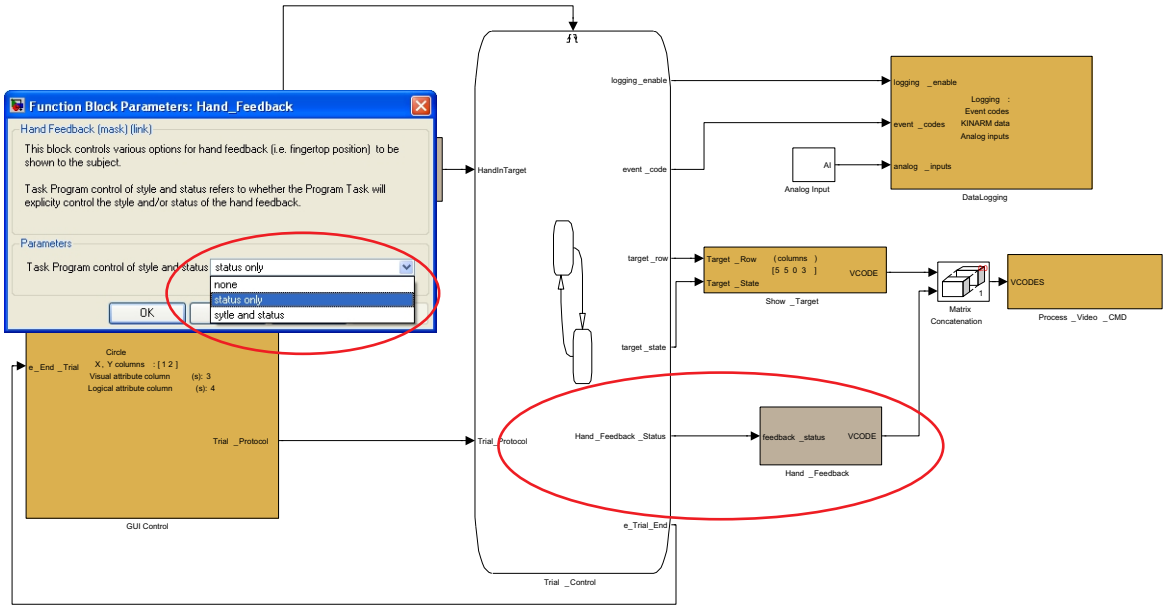
6.12 Controlling Hand Feedback

The purpose of this example is to demonstrate how visual hand feedback can be controlled on-line during a task (e.g. on for some trials and off for others, or changing on/off part-way through a trial). In order to understand what will actually occur during a task, it is important to realize that hand feedback control in BKIN Dexterit-E is multi-tiered. BKIN Dexterit-E allows an end-user to choose the control of hand feedback either as on, off or controlled by the Task Program. The settings chosen within BKIN Dexterit-E will override those chosen within the Task Program, so the Task Program can only control hand feedback if the end-user has chosen “Task Program Control” for hand feedback within BKIN Dexterit-E (see the Dexterit-E User Guide for more information).

Simulink Code for Example 6.12

From within the `Hand_Feedback` block, there are various options for choosing hand feedback control. In this example, Task Program control of “status only” has been chosen, which means that the status (i.e. on/off state) of hand feedback can be controlled by the Task Program through the feedback_status input. For more information on hand feedback control options, click on the help button of the `Hand_Feedback` block within Simulink.

Fig 6-12. Simulink Code for Example 6.12 - "Controlling Hand Feedback"



Stateflow Chart for Example 6.12

The Stateflow Chart for this example is not shown here. The main feature is a new output `Hand_Feedback_Status` which must be set as desired from within stateflow. (e.g. `Hand_Feedback_Status=1`)

6.13 Error Trials: Repeating and/or Reporting

The purpose of this example is to demonstrate how an error trial can be forced to be repeated and/or recorded. In this context, an error trial is a trial in which the subject has done something incorrectly (e.g. did

not reach to a target). In this example, we set up the task to both repeat and record the error, however, only one or the other need be done. In other words they are independent from each other, so if both types of functionality are desired, then both need to be implemented.

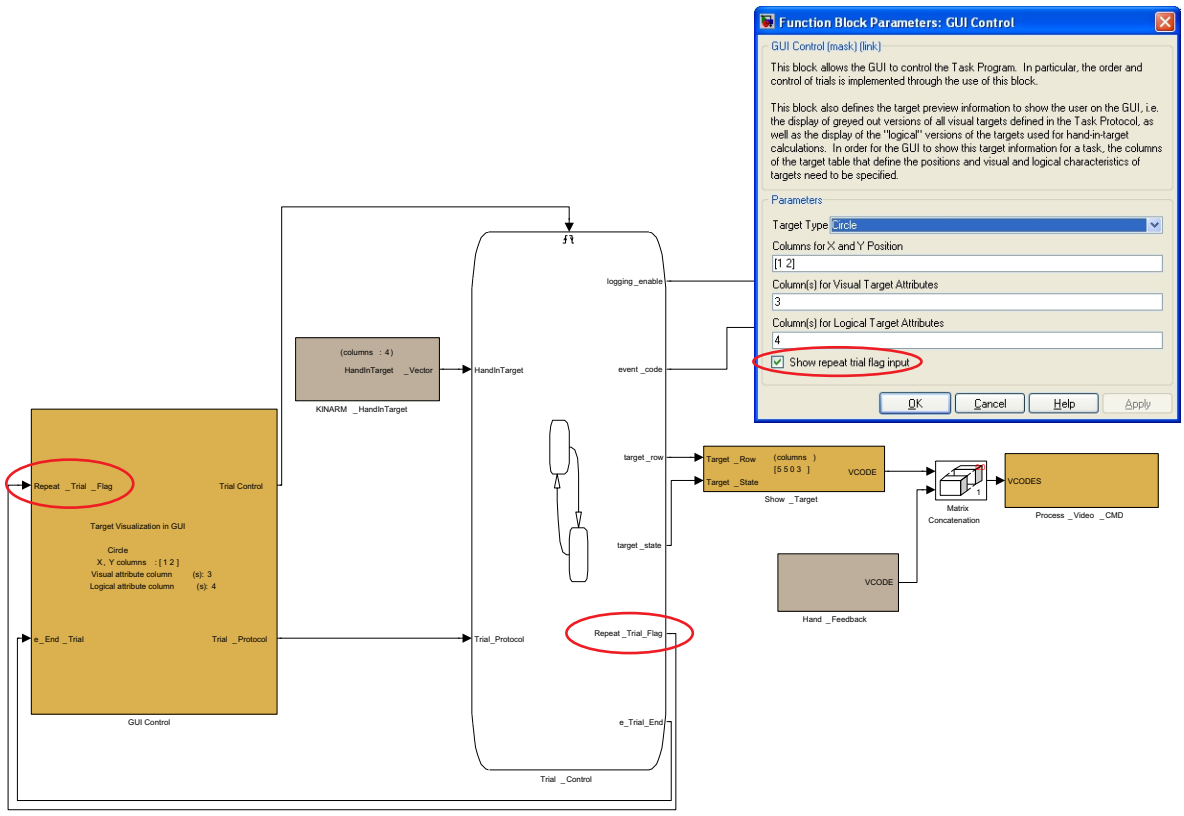
In order to potentially repeat the trial (which will occur at the end of the current block), the `Repeat_Trial_Flag` input of the GUI Control block is used. This input is only available if the `Show repeat trial flag` input checkbox has been checked in the `GUI Control` dialog. Its purpose being to allow the option for a trial to be repeated if desired. However, in order to understand what will actually occur during a task, it is important to realize that Repeat Error Trials control in BKIN Dexterit-E is multi-tiered. BKIN Dexterit-E allows an end-user to choose whether or not the `Repeat_Trial_Flag` is ignored. If the user selects “Repeat Error Trials” from within BKIN Dexterit-E, then if the `Repeat_Trial_Flag>0` at the end of a trial, that trial will get repeated at the end of the block. If the user does not select “Repeat Error Trials”, or if `Repeat_Trial_Flag=0`, then the trial will not get repeated. If the nothing is wired into the Repeat_Trial_Flag, then it is always =0 and no trials ever get repeated.

In order to report a trial as being an error trial to the BKIN Dexterit-E User Interface, an event has to be created to report the error and that event needs to be defined as an error-reporting event. Multiple error-reporting events can be defined for a single task. If any error-reporting event occurs during a trial, BKIN Dexterit-E records that trial as an error-trial and the % correct statistics in the BKIN Dexterit-E window are updated appropriately.

Simulink Code for Example 6.13

In this example, a connection is made from the Stateflow chart to the `Repeat_Trial_Flag` input of the `GUI Control` block, after checking the Show repeat trial flag input checkbox.

Fig 6-13. Simulink Code for Example 6.13 - "Error Trials: Repeating and/or Reporting"

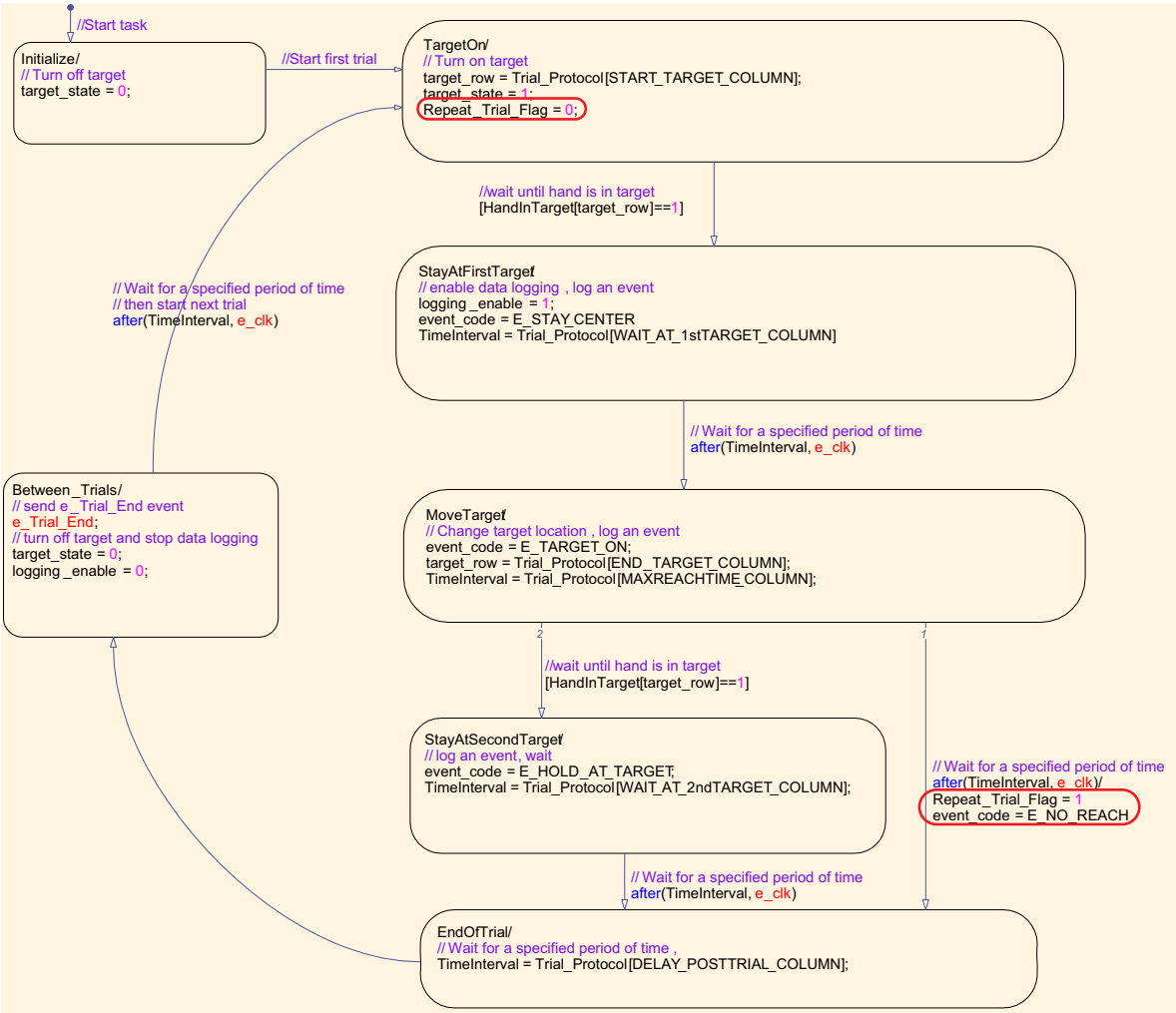


Stateflow Chart for Example 6.13

In order to repeat an error trial, the new Stateflow output `Repeat_Trial_Flag` must be set as desired from within stateflow (e.g. if something does not occur as desired, then set `Repeat_Trial_Flag=1`). In the example shown below, the `Repeat_Trial_Flag` is set =0 at the start each trial. If the subject does not reach to the second target before a given time has expired (i.e. from `tickCount = Trial_Protocol[MAXREACHTIME_COL]`), then `Repeat_Trial_Flag=1` is set.

In this example if the subject does not reach to the second target before a given time has expired then `event_code=E_NO_REACH` is also set. This event code will be stored as part of the trial's data file and so can be used by subsequent data analysis to indicate that this trial was an error trial. In addition, in this example we have defined this event code as an error-trial event code in `<your_task>_cfg.xml`. When an event code is defined as an error-trial event code, it means that this trial will be reported to BKIN Dexterit-E as an error trial. BKIN Dexterit-E uses error-trial event codes to update task statistics (i.e. percent trials correct).

Fig 6-14. Stateflow Code for Example 6.13 - "Error Trials: Repeating and/or Reporting"



`<your_task>_cfg.xml` for Example 6.13

For an event code to report to BKIN Dexterit-E that this trial was an error trial, the event code definition must include a `ui_event` tag as shown below. This tag is an optional tag for event codes which allows a particular event code (or multiple event codes) to be flagged as an indication of an error.

```
<Event code="99" name=”NO_REACH" desc="No reach occurred" ui_event="error"/>
```

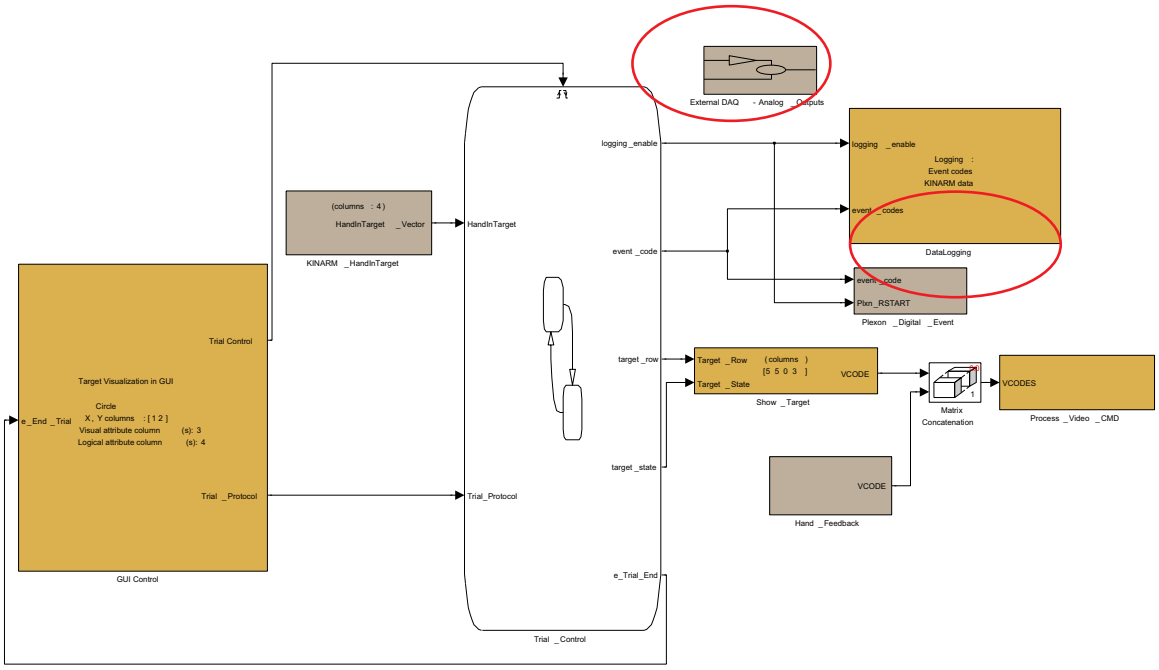
6.14 External DAQ

The purpose of this example is to show what changes are necessary to have a Task Program function with an external data acquisition system. The purpose of such a setup is to have a central data acquisition system rather than multiple systems which would require post-experiment synchronization of data files (e.g. if using a neural recording system such as Plexon). This type of setup requires different hardware than the standard BKIN Dexterit-E requirements. Refer to the Dexterit-E User Guide for more information.

Simulink Code for Example 6.14

In this example, two new Simulink blocks are required to send information over to the external data acquisition system. Analog information relating to KINARM robot kinematics is sent using the `External DAQ-Analog_Outputs` block (i.e. joint position, velocity and acceleration) while user-defined event code related information is sent using the `Plexon_Digital_Event` block (this is a Plexon-specific example). For more information on interfacing to an external DAQ system with these Simulink blocks, please click the Help button of those blocks, available from within Simulink.

Fig 6-15. Simulink Code for Example 6.14 - "External DAQ"



Note: The reason for sending velocity and acceleration data to an external system in addition to position is that the conversion from digital to analog (in BKIN Dexterit-E's real-time computer) and then back from analog to digital (in the external DAQ system) introduces quantization error into the position signal. This quantization error is very small (<0.05% for standard 12-bit systems) but it becomes magnified upon subsequent differentiation. Refer to the Dexterit-E User Guide for more information.

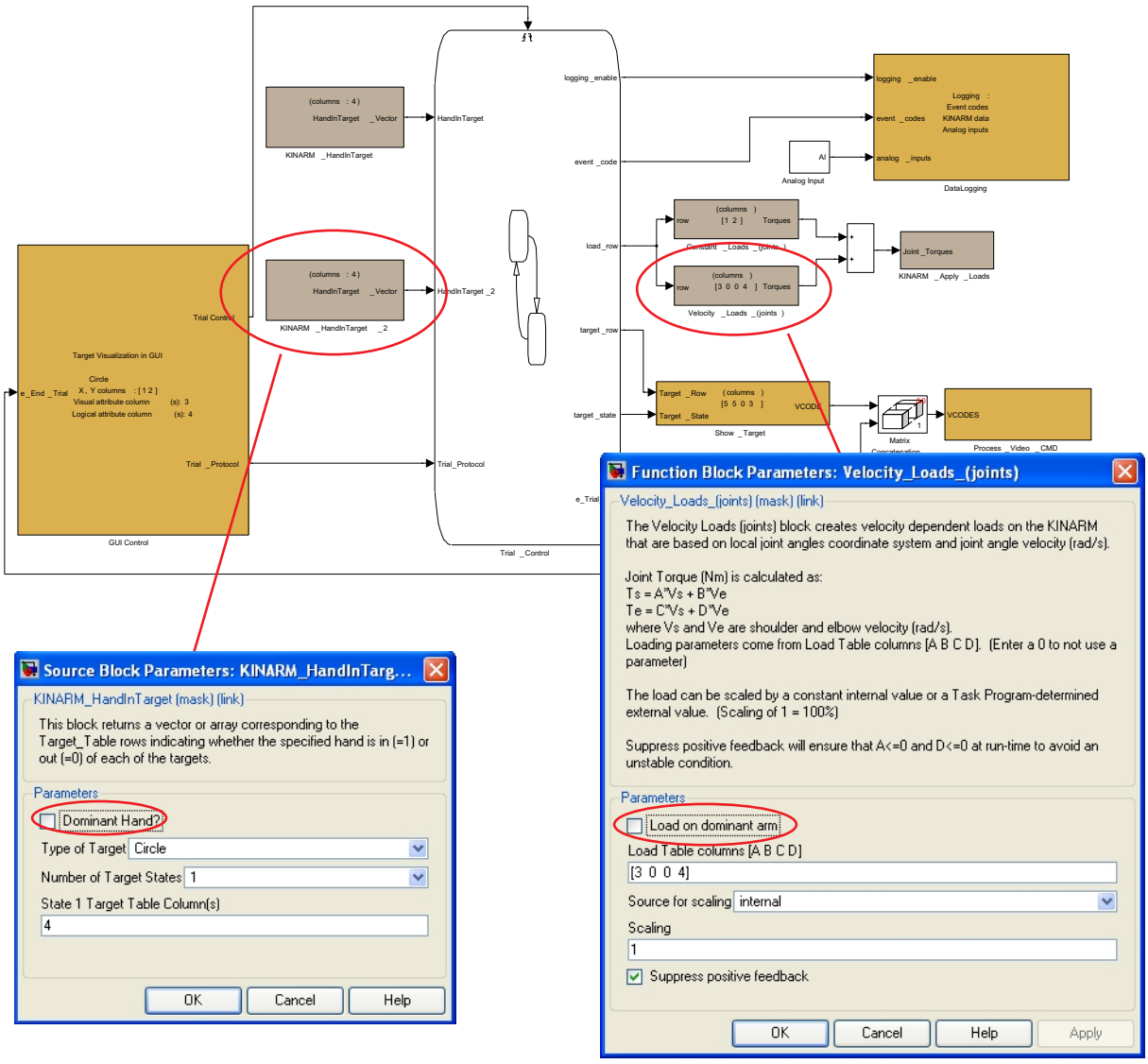
6.15 Bilateral KINARM Lab Feedback and Loads

The purpose of this example is to show some basics of bilateral KINARM task control, including feedback about hand position for both arms, and loading conditions on both arms.

Simulink Code for Example 6.15

In this example, there are now two copies of the `KINARM_HandInTarget` block (as compared to Example 6.5 - "Loads on the KINARM Robot"). Furthermore, as explained in more detail below, the parameters for blocks circled below have been changed.

Fig 6-16. Simulink Code for Example 6.15 - "Bilateral KINARM Lab Feedback and Loads"

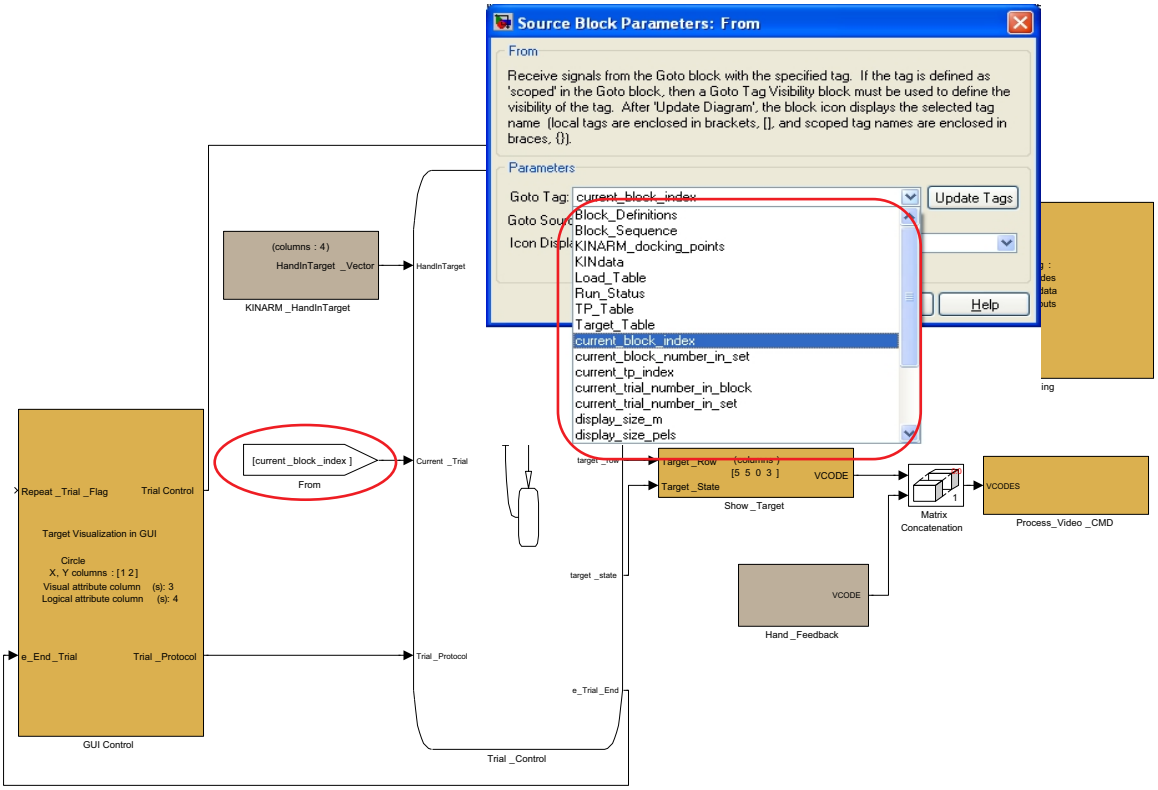


The only difference between the two circled blocks and their un-circled counterparts is the unchecked box for “Dominant Hand”. When a Task Program and Task Protocol are chosen in BKIN Dexterit-E, the user has the option of choosing the task’s dominant arm to be the right arm or the left arm. The meaning of dominant arm is Task Program specific, as defined by blocks such as these. The non-dominant hand is thus the other hand. Task Programs in Simulink are therefore not defined in terms of right-hand or left-handed, but rather in terms of a “dominant” and “non-dominant” hand. If the user chooses a Task Protocol in BKIN Dexterit-E for the “right arm”, that means that the right arm is the dominant arm, so `KINARM_HandInTarget` is going to output what the right arm is doing, while `KINARM_HandInTarget_2` is going to output what the left arm is doing (i.e. the non-dominant arm). If the user chooses a Task Protocol for this same Task Program, but the selects the “left arm”, then the left arm will the dominant arm and `KINARM_HandInTarget` is going to output what the left arm is doing, while `KINARM_HandInTarget_2` is going to output what the right arm is doing (i.e. the non-dominant arm).

6.16 Accessing KINdata, Current Block Index, and Other Task Control Variables

It is possible to access various task control variables used by the Simulink blocks that make up the BKIN Dexterit-E library through the use of the Simulink's `From` block (see Simulink\Signal Routing library). The `From` block creates an invisible connection to a `Goto` block elsewhere in the Simulink model (i.e. using `Goto` and `From` blocks is the same as adding a connection wire from the source to the destination, but without the clutter of a visible wire; multiple `From` blocks can access a single `Goto` block). Accessing various task control variables can be desirable, for example, if something needs to be done only on the first trial (in which `current_trial_number_in_set` would be a useful variable to access) or if a custom block is being created that requires access to various KINARM data (in which case `KINdata` would be the relevant variable to access). Paste the `From` block into a model, double-click it and select one of the `Goto Tag` options to see a list of the available parameters that can be read from (make sure to first click `Update Tags` so that the list is complete). These blocks allow the end user to access things such as the various parameters tables (e.g. `Target_Table`) as well as status of the overall task (e.g. `current_block_index`, `current_tp_index`). For more information regarding which Goto tags are available, please see Section 9.7 - "Available 'Tags' (From and Goto Blocks)".

Fig 6-17. Simulink Chart for Example 6.16 - "Accessing KINdata, Current Block Index, and Other Task Control Variables"



Note: To find out more about what each of the possible variables are, choose the relevant Goto tag, then click Goto Source in the From block dialog to open the Simulink block in which the Goto block resides.

6.17 Multiple Conditions for Stateflow Transitions (and e_clk event)

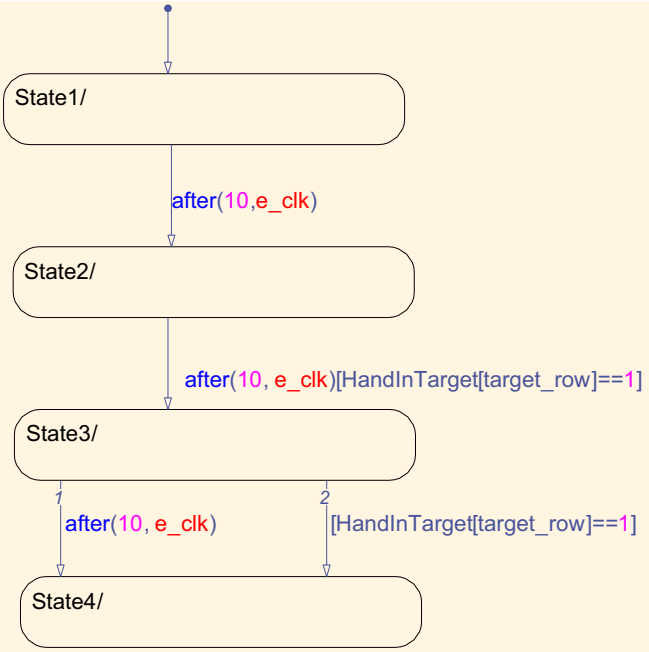
The purpose of this example is to demonstrate a few of the more common transition conditions that end-users are likely to use and how to combine multiple conditions appropriately. For more information on transitions, please refer to Mathworks' Stateflow documentation.

Stateflow Chart for Example 6.17

In the example shown here, we have assumed that the `e_clk` event is defined in Stateflow's Model Explorer as per the sample Task Programs included with BKIN Dexterit-E. Namely, `e_clk` is the first input Stateflow event and is triggered only on the rising edge. In this manner, `e_clk` events in a Task Program will occur once every 1 ms.

For the Stateflow chart below, the transition from `State1` to `State2` occurs after 10 e_clk events have occurred (i.e 10 ms). The transition from `State2` to `State3` occurs only if the `HandInTarget` condition is true AND at least 10 `e_clk` events have occurred (i.e. both conditions must be true). The transition from `State3` to `State4` occurs if the `HandInTarget` condition is true OR 10 `e_clk` events have occurred (i.e. whichever condition occurs first).

Fig 6-18. Stateflow Chart for Example 6.17 - "Multiple Conditions for Stateflow Transitions (and e_clk event)"



6.18 Random Numbers in Stateflow

Creating random numbers in Stateflow requires using the C notation as shown in the example below. The function `rand()` returns an integer between 0 and `RAND_MAX`. The constant `RAND_MAX` is an integer constant defined by the C library being used. It is typically 32767, although it can be much higher. The function `srand()`, which creates the seed used by the `rand()` function, is called within the `GUI Control` block at the start of every task and so it is not required to by added by the end-user.

To create a random number between 0 and 1, the following line can be added to a Stateflow chart:

```
random_number = rand()/RAND_MAX;
```

To ensure that `rand()` and `RAND_MAX` are available:

1. In the Stateflow chart window, choose Tools → Open Simulation Target.
2. Click the Custom Code tab.
3. In the Include Code tab, add the following: `#include <stdlib.h>`
4. Click Apply and OK to save this change and exit.

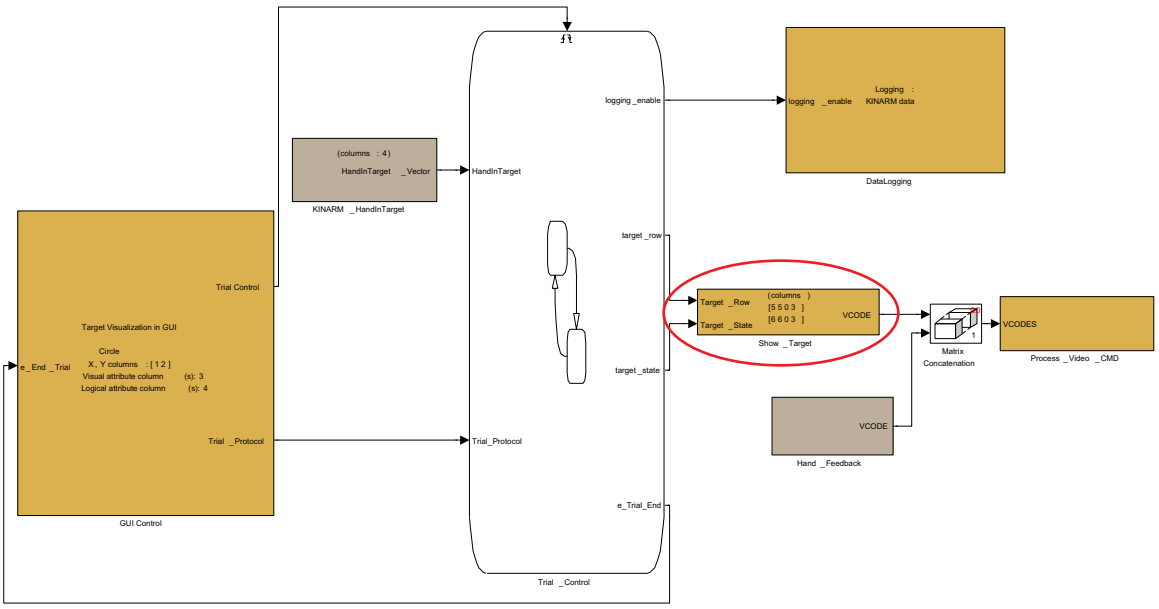
6.19 Parallel State Execution (Independent State Machines)

The purpose of this example is to show how to set up independent state machines that operate in parallel. In this example, control over the target (i.e. target color) is controlled independently from the main flow of states in the task.

Simulink Code for Example 6.19

The Simulink code for this example is based on the code in Example 3.5 - "A Basic Task Program". The one difference is the addition of a second target state in the `Show_Target` block. Please see Example 6.7 - "Multiple Targets and Multiple Target States" for more details on creating a second target state.

Fig 6-19. Simulink Code for Example 6.19 - "Parallel State Execution (Independent State Machines)"



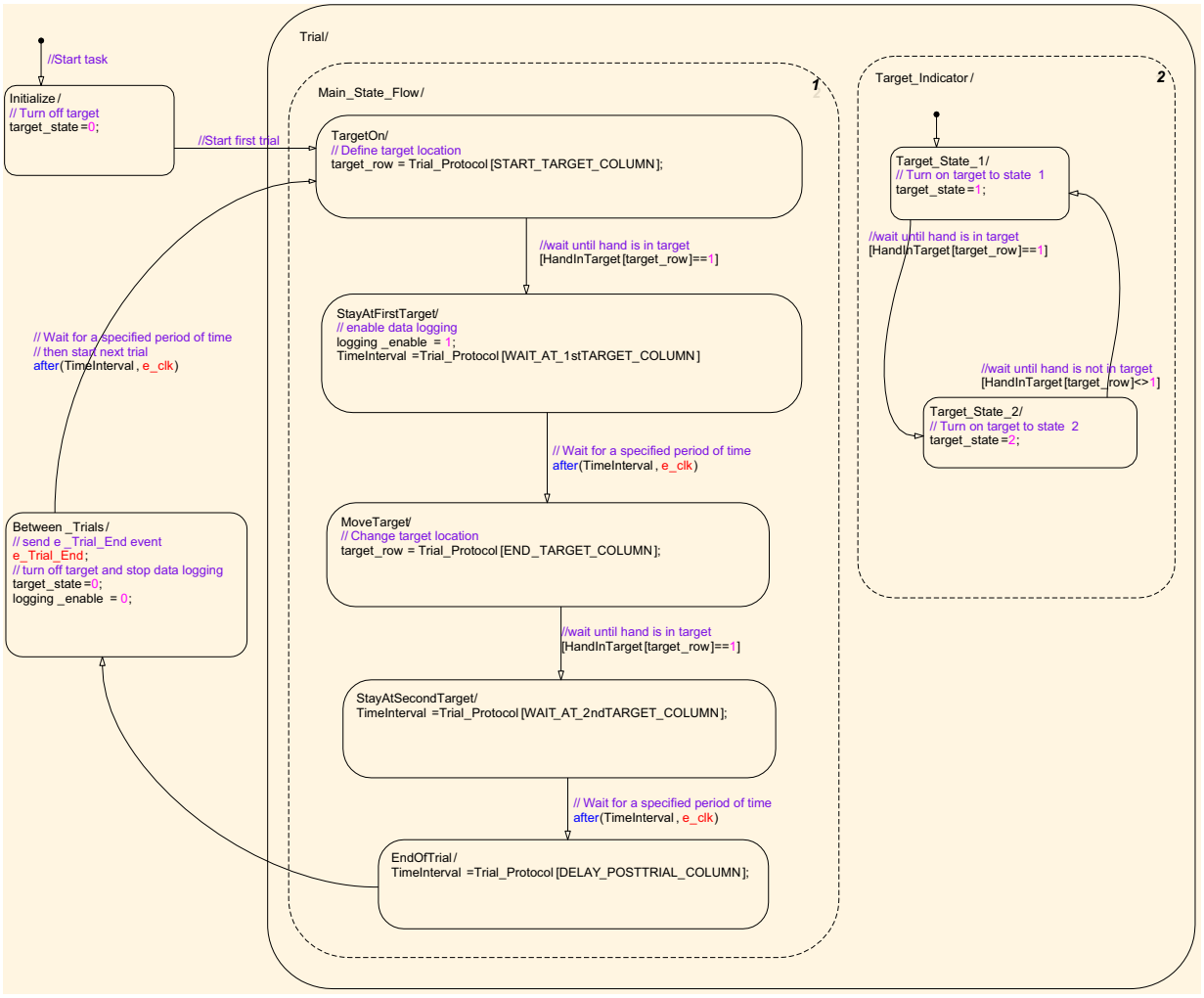
Stateflow Chart for Example 6.19

The Stateflow Chart shown below is based upon Example 3.5 - "A Basic Task Program", however, several modifications were made to the chart:

1. A new super-state, `Main_State_Flow`, was created encompassing most of the original states

2. A new state, `Target_Indicator`, was created along with two new sub-states, `Target_State_1` and `Target_State_2`.
3. A new super-state, `Trial`, was created that encompassed the `Main_State_Flow` and `Target_Indicator` states.
4. The decomposition of the super-state `Trial` was changed to Parallel (AND) by right-clicking inside of the super-state and choosing Decomposition→ Parallel (AND). This step resulted in the `Main_State_Flow` and `Target_Indicator` states becoming dashed lines rather than solid lines to indicate that they will operate concurrently. The number in the upper right corner indicates which will execute first.

Fig 6-20. Stateflow Chart for Example 6.19 - "Parallel State Execution (Independent State Machines)"



The above modifications mean that when a transition occurs into the `Trial` super-state, both the `Main_State_Flow` and `Target_Indicator` sub-states become active and both will operate and function concurrently. Likewise, when a transition occurs out of the `Trial` super-state, both the `Main_State_Flow` and `Target_Indicator` super-states become inactive and both cease to operate. For more complete information on using Parallel versus Exclusive states in Stateflow, please refer to Mathworks' Help documentation.

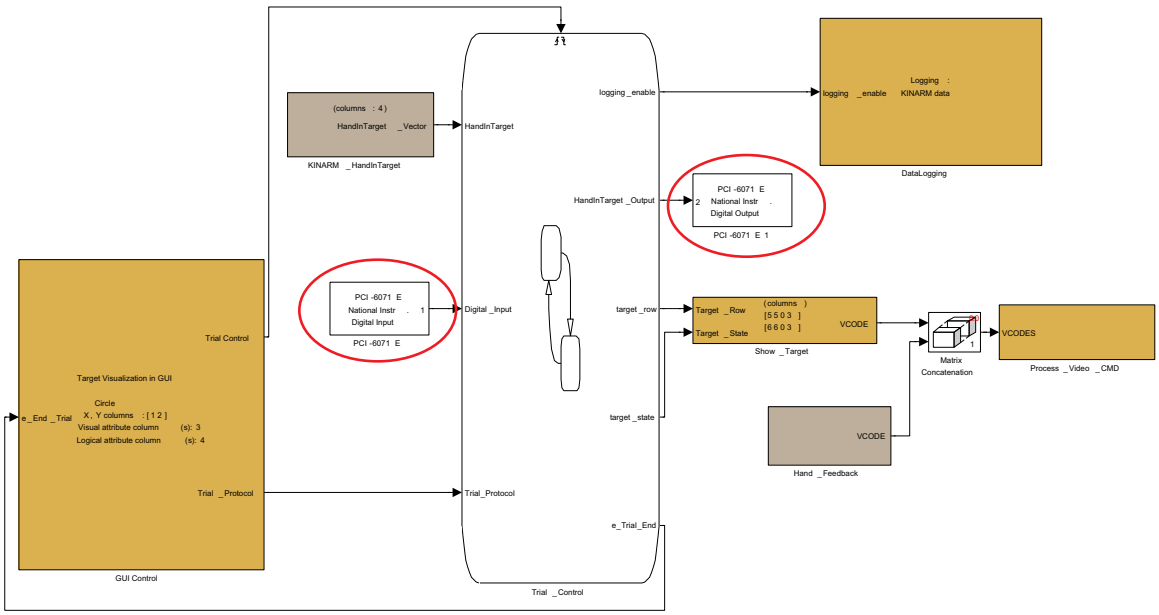
6.20 Digital Input/Output

The purpose of this example is to show how to set up digital inputs and outputs. In this example, a digital input is used to provide additional control over when a task transitions from one state to the next, and a digital output is used to indicate when a subject's hand is in or out of a target. The digital input could be from a push-button that the experimenter has and the digital output could go to a speaker to provide audio output.

Simulink Code for Example 6.20

The Simulink code for this example is based on the code in Example 3.5 - "A Basic Task Program". The differences are the addition a Digital Input block and a Digital Output block, as well as the associated inputs/outputs in `Trial_Control`.

Fig 6-21. Simulink Code for Example 6.20 - "Digital Input/Output"

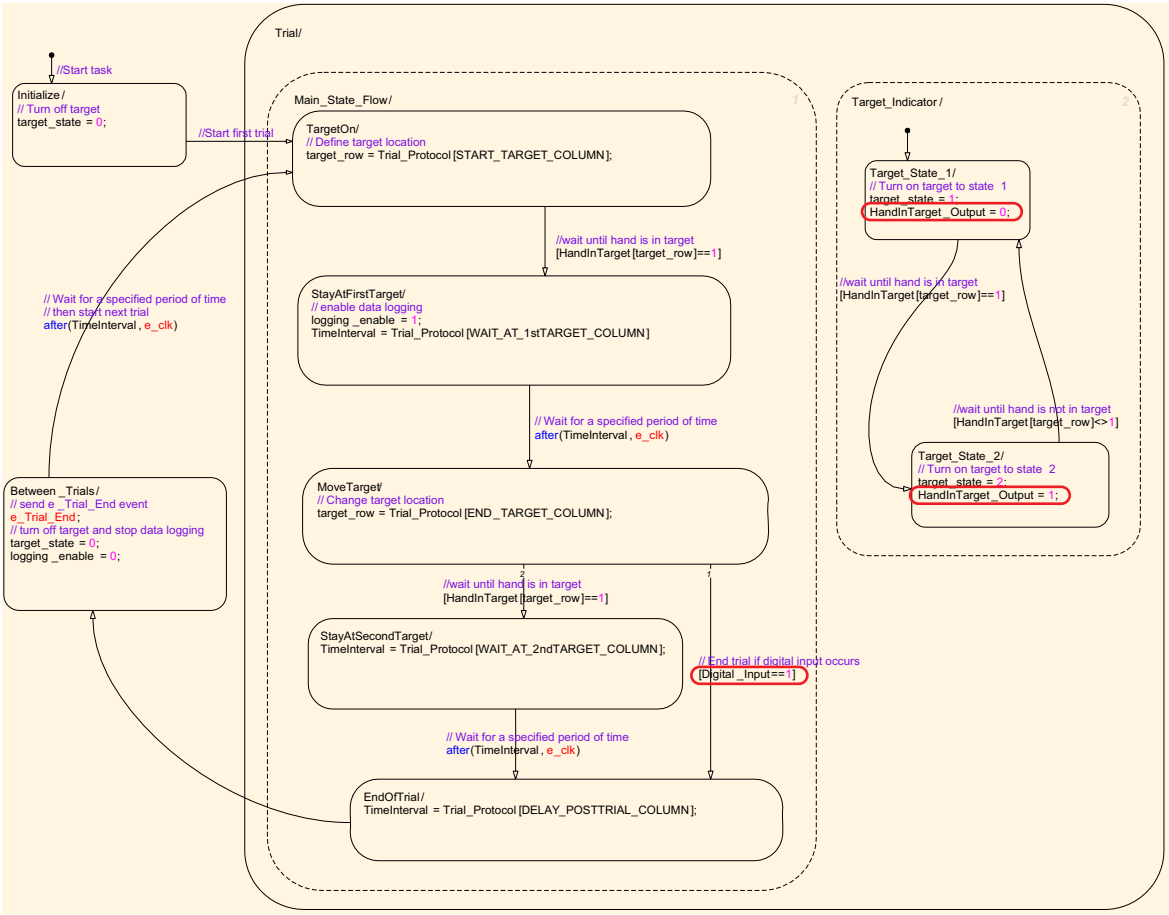


Stateflow Chart for Example 6.20

The Stateflow Chart shown below is based upon Example 6.19 - "Parallel State Execution (Independent State Machines)", with a few modifications:

1. A new transition from `MoveTarget` to `EndOfTrial` was added, based on the condition `[Digital_Input==1]`. Thus if the push button connected to digital input 1 is pressed when the task is in the MoveTarget state, then `[Digital_Input==1]` will become true and the transition will occur.
2. `HandInTarget_Output` is a newly defined output whose value depends on the `HandInTarget` value. If `[HandInTarget==1]`, then the Task Program will set `HandInTarget_Output=1`, which in this example is connected to a digital output that goes to a speaker, thus providing audio feedback on when a subject's hand is in the desired target.

Fig 6-22. Stateflow Chart for Example 6.20 - "Digital Input/Output"



6.21 Synchronization of BKIN Dexterit-E Data with External Clock

The purpose of this example is to demonstrate how to provide a synchronization pulse that can be used to synchronize data collected by two independent data collection systems. Consider, for example, a system in which kinematic data were to be saved by BKIN Dexterit-E while simultaneously an external, independent data collection system were to save EEG data. In order to analyze relationships between the kinematic and EEG data, there would have to be a way to synchronize the data to be sure the events in one data set were related to events in the other.

The problem of synchronization is not necessarily an obvious one. It originates because no two clocks run at exactly the same rate. Two systems that both run nominally at 1 kHz, will always exhibit some amount of drift over time – i.e. one clock will always be slower or faster than the other. In some situations, the amount of drift will be negligible, and so synchronization between two data sets becomes trivial – if both data collection systems start on the same trigger, then the data will be synchronized. However, in most real-life situations, the amount of drift is large enough to be noticeable, and so it must be accounted for.

Many data collection systems avoid the problem of synchronizing multiple data sets by using a single master clock that drives all data collection systems (i.e. synchronization is forced by using only a single clock). This approach requires (a) that all but one of the data collections systems be capable of being driven by a master clock and (b) that the master clock pulse timing is precise and reliable. Many hardware systems do not meet these criteria, and so the problem of synchronizing two independent data sets remains. The example here shows one potential method of addressing this problem.

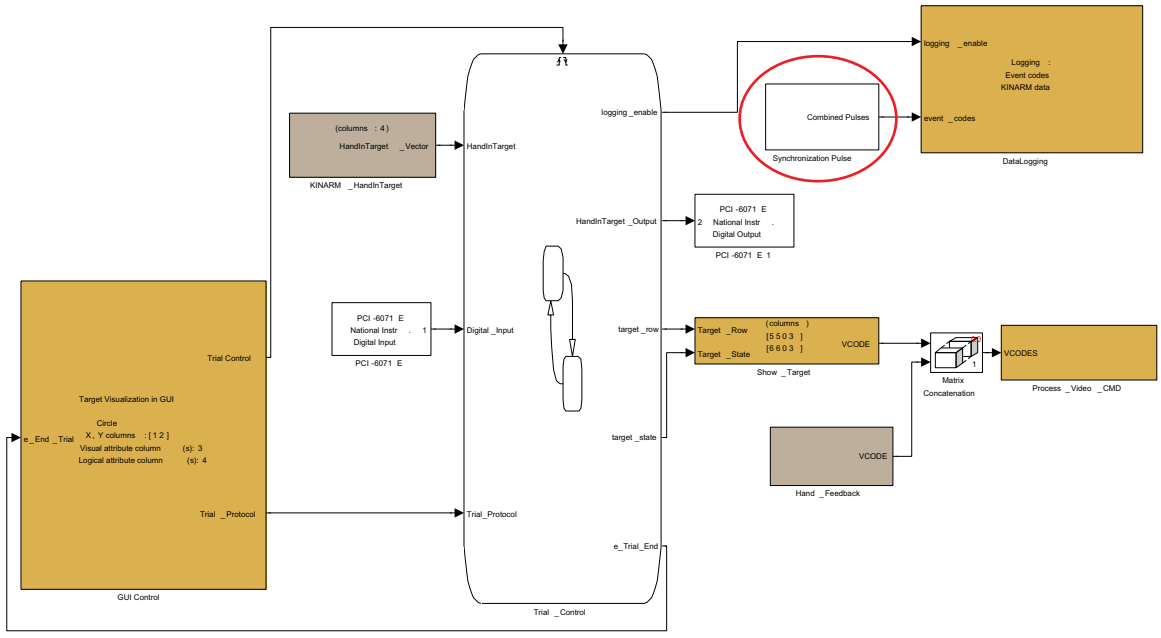
The solution demonstrated here is to have the Task Program controlled by BKIN Dexterit-E send out clock signals at a known rate that are then recorded in real-time by the second system. The amount of synchronization that will need to be done (i.e. synchronize once per trial, versus once per second) will depend on the length of the trials to be collected, the required accuracy and relative drift of the two clocks. In this example, a single synchronization pulse is sent out digitally at the beginning of the trial and additional pulses are sent every 1000 ms (i.e. synchronization between the clocks is done at the beginning of each trial and then every second subsequently).

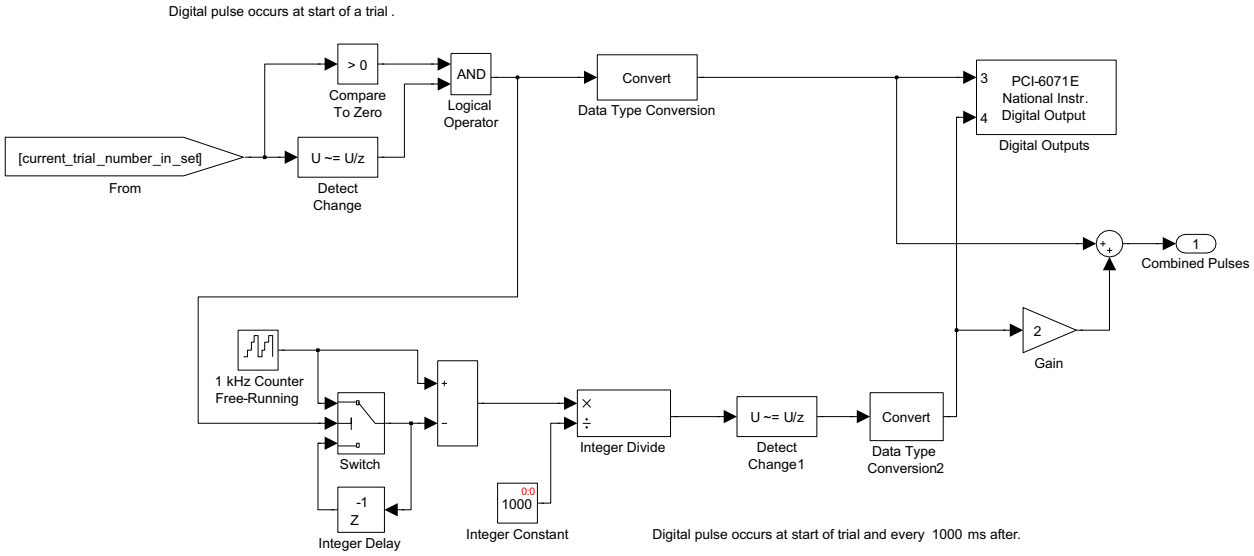
Simulink Code for Example 6.21

In this example, which is based upon Example 6.20 - "Digital Input/Output" a new block called `Synchronization Pulse` has been created. Within the `Synchronization Pulse` block are two sections of code, one that creates a pulse output every time a trial starts and the other which creates a pulse every 1000 ms. The first section of code is based around a `From` block which is connected to the `current_trial` tag. The `current_trial` tag is a counter of the current trial number within a block. Between blocks the `current_trial` variable is reset to 0 and is incremented at the start of each trial. So it is equal to 1 for the first trial, 2 for the second trial etc. If a task is paused between trials, it retains the value of the last trial, with the exception that if a task happens to be paused between blocks, then `current_trial=0`. The second section of code is based around a free-running 1 kHz counter. The value of the free-running counter at the start of the trial is stored and subtracted from the ongoing value of the free-running counter to create an secondary counter that is initialized to zero at the beginning of each trial. This secondary counter is then monitored to create pulses every 1000 ms after the start of the trial.

Note:For more information on accessing current_trial and other task control variables, please see Example 6.16 - "Accessing KINdata, Current Block Index, and Other Task Control Variables".

Fig 6-23. Simulink Code for Example 6.21 - "Synchronization of BKIN Dexterit-E Data with External Clock"

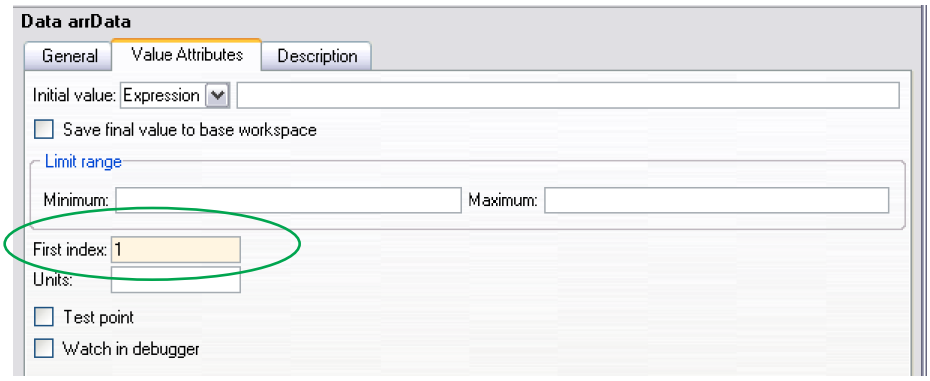




6.22 Using Vectors in Stateflow

If you wish to use an vector in your stateflow chart it is essential that if it is a local variable you set the vector's size, but no matter what you must always state its starting index. Users familiar with Matlab will be well aware that Matlab uses 1 based indexing, however, the default in stateflow is 0 based. This can be set within the Model Explorer. Select the vector you wish to work with and look at its properties. On the Value attributes tab be sure to set the First index field.

Fig 6-24. Defining the first index of an array in Model Explorer



7 BKIN Dexterit-E Task Development Kit Libraries

All of the Simulink blocks contained within the various libraries provided as part of BKIN Dexterit-E Task Development Kit have built-in help files to describe in detail how the block should be used (help files are accessed through Simulink). Those help files are not replicated here, but rather a broad description of each of the libraries is provided to understand the organization of the various blocks.

7.1 General

This library contains general blocks for use with Task Programs. Examples include:

- `GUI Control` (required for communications with the BKIN Dexterit-E GUI)
- `DataLogging` (required for communications with the BKIN Dexterit-E GUI)

7.2 KINARM General

This library contains general blocks related to the KINARM Lab. Examples include:

- `Poll_KINARM` (automatically embedded in `DataLogging` block if logging of KINARM data is checked)
- `Hand_Feedback` (required if hand feedback display to the subject is desired)
- `HandInTarget` (provides feedback indicating which targets in the target table one of the hands is in)
- `DistanceFromTarget` (provides feedback indicating the distance between a targets and one of the hands)

7.3 KINARM I/O

This library contains input/output blocks related to the KINARM Lab (e.g. motion control card, analog and digital I/O). Examples include:

- `External_DAQ-Analog_Outputs` (outputs KINARM robot kinematic information over analog channels for recording by an external data acquisition system)
- `External_DAQ-Digital_Outputs` (outputs two digital signals over cables labelled “Reward” and “Record” - typical use is for the reward signal and to trigger an external system to begin recording (e.g. eye-tracking system), but they can be used for anything)
- `Internal_DAQ-clock_Output` (outputs 1 kHz clock from the real-time computer over a digital channel when BKIN Dexterit-E is performing data acquisition)
- `Plexon_Digital_Event` (outputs user-defined event codes to Plexon and also controls when Plexon records data)

7.4 KINARM Loads

This library contains blocks for applying loads with the KINARM Exoskeleton robot. Examples include:

- `KINARM_apply_loads` (required for applying a load to the KINARM robot)
- `Constant_Loads_(hand)` (applies a constant load in a hand-based coordinate frame)
- `Constant_Loads_(joints)` (applies a constant load in joint-based coordinate frame)
- `Velocity_Loads_(hand)` (applies velocity-based loads in a hand-based coordinate frame)
- `Velocity_Loads_(joints)` (applies velocity-based loads in joint-based coordinate frame)

- `Velocity_Loads_(hand, inter-arm)` (applies velocity-based loads in a hand-based coordinate frame, where load on one arm depends on velocity of other arm)
- `Velocity_Loads_(joints, inter-arm)` (applies velocity-based loads in joint-based coordinate frame, where load on one arm depends on velocity of other arm)
- `Forces_to_Torques` (converts forces in a hand-based coordinate frame to torques in a joint-based coordinate frame)
- `Perturbation` (creates a time-dependent profile that can be used as the scaling input to another load block to create a perturbation)

7.5 KINARM EP Loads

This library contains blocks for applying loads with the KINARM Endpoint robot. Examples include:

- `KINARM_EP_Apply_Loads` (required for applying a load to the KINARM robot)
- `Constant_Loads_EP_(hand)` (applies a constant load in a hand-based coordinate frame)
- `Perturbation_EP` (creates a time-dependent profile that can be used as the scaling input to another load block to create a perturbation)
- `Velocity_Loads_EP_(hand)` (applies velocity-based loads in a hand-based coordinate frame)
- `Velocity_Loads_EP_(hand, inter-arm)` (applies velocity-based loads in a hand-based coordinate frame, where load on one arm depends on velocity of other arm)

7.6 Video

This library contains blocks used to display video. Examples include:

- `Show_Target` (creates a `VCODE` containing all target information based on the Target Table and target selection)
- Show_Target_With_Label (creates a VCODE containing all target information based on the Target Table and target selection for targets with text).
- Set_Target_in_Background (takes a VCODE and turns it into a target definition that does not need to constantly be transmitted once its state is set)
- `Process_Video_CMD` (processes `VCODEs` and communicates with computer that actually displays the video)

8 Custom Simulink Blocks and Libraries

In many cases, end-users will want or need to create their own custom Simulink block to implement a novel feature. Furthermore, end-users may want such a block to exist in their own custom library to make it available for multiple Task Programs. This chapter provides basic instructions on creating custom Simulink blocks and adding them to a custom Simulink library.

8.1 Creating a Custom Simulink Block

For end-users wishing to create their own customized Simulink blocks, we recommend that they begin by looking at the blocks provided in the libraries as part of BKIN Dexterit-E. Many of the blocks provided with BKIN Dexterit-E include all of the Simulink and Matlab source code which provides an excellent 'tutorial' for how to program. To see this source code:

1. Drag a block from one of BKIN Dexterit-E's Simulink libraries to a Task Program. Make sure to choose a block that is similar to what you are intending to do with your custom block.
2. Right-click on the block and under "Link Options" choose Disable Link (this disables the link between the library and this instance of the block).
3. Right-click on the block and choose "Look Under Mask". This opens the Simulink block.

8.2 Putting Your Custom Block Into a Custom Library

If you intend on using a custom block in more than one Task Program, it is advantageous to put that custom block into a Simulink library. The following steps will enable you to create and see your own custom library.

1. Go to New→ Library from within Simulink to create a new library. Save it somewhere appropriate with an appropriate name.
2. To have your custom library show up in the Simulink Library Browser, you need an `slblocks.m` file in the same directory that your custom library was just saved in. Search for existing ones in the Simulink directory and modify as per instructions in the `slblocks.m` file.
3. Add the path of the directory containing the new library to the Matlab Path (note: you will also have to close the Simulink Library Browser and restart it to have your new library show up).

If you want only your custom block to show up in the library (and not any of the sub-blocks within your custom block), then you will need to Mask your block (see Mathworks' documentation).

If you want that library to have sub-libraries, add subsystems to the main custom library and enter your custom blocks into the subsystems.

If you want your custom library to have sub-libraries that are separate library files (as per BKIN Dexterit-E's library structure) rather than contained within sub-systems, then the sub-systems in the main library should be empty, and then in each of the subsystems' `Block_Parameters`, go to `CallBacks` and the `OpenFcn` -- add in the name of the sub-library here (should be in the same directory as the main library).

9 Reference

This chapter contains reference information that is useful for creating Task Programs.

9.1 Task File Types: .mdl, .dlm, .dtp and .xml

There are four main file types associated with a task:

.mdl – `<your_task>.mdl` files are Simulink model files that contain the source code for Task Programs. These files are used by Matlab and Simulink when creating and building a custom Task Program. These files are not used directly by BKIN Dexterit-E. This file is created within the Simulink environment.

.dlm – `<your_task>.dlm` files are the compiled, or "built", versions of Task Programs. These files are used by BKIN Dexterit-E when running a task. This file is created by Simulink as the result of a build.

.dtp – `<your_task_protocol>.dtp` files are text files that contain all details for a Task Protocol. These files are used by BKIN Dexterit-E when running a task and must reside in the same directory as `<your_task>.dlm`. These files are created within BKIN Dexterit-E.

.xml – `<your_task>_cfg.xml` files are XML files that contain the Task Protocol parameter table definitions for a given Task Program. These files are used by BKIN Dexterit-E when running a task and must reside in the same directory as `<your_task>.dlm`. These files are created manually using a text editor.

Note: The build process creates a .xml file, called <your_task>.xml. This is not the same .xml file. Because Simulink always creates a .xml file during the build process, and because BKIN Dexterit-E uses the first .xml file it finds in a task directory, we recommend not building <your_task>.mdl in the same directory that BKIN Dexterit-E is looking for <your_task>.dlm.

9.2 Structure of KINdata

The read-only KINdata matrix, which can be accessed as shown in Example 6.16 - "Accessing KINdata, Current Block Index, and Other Task Control Variables", contains all relevant kinematic data related to the KINARM robot. It allows end-users to create custom effects based on feedback from the KINARM robot. Within the KINdata matrix, row 1 contains data about the first arm and row 2 contains data about the second arm, as shown in **Table 9-1**. For bilateral KINARM Labs, the first arm is the right arm and the second arm is the left arm. For unilateral systems, the first arm can be either the right or left arm, depending upon the KINARM configuration.

Table 9-1: Arm specific data in KINdata matrix (i.e. KINdata(i,j), where i=1 or i=2)

j	Name	Description
1	L1	Length (m) of KINARM segment L1, the upper arm (shoulder to calibrated elbow position)
2	L2	Length (m) of KINARM segment L2, the forearm (calibrated elbow position to calibrated fingertip position)
3	L2_ptr	Location (m) of fingertip, relative to KINARM segment 2. (Measured in the anterior direction when the arm is in the anatomical position, i.e. when the arm is straight out to the side)
4	SHO_X	Shoulder position (m), x-axis in global coordinate scheme.
5	SHO_Y	Shoulder position (m), y-axis in global coordinate scheme.

Table 9-1: Arm specific data in KINdata matrix (i.e. KINdata(i,j), where i=1 or i=2)

j	Name	Description
6	ORIENTATION	1= right-handed KINARM robot, 2 = left-handed KINARM robot
7	SHO_ANG	Shoulder angle (rad) in local coordinates (flexion is positive)
8	ELB_ANG	Elbow angle (rad) in local coordinates (flexion is positive)
9	SHO_VEL	Shoulder angular velocity (rad/s) in local coordinates
10	ELB_VEL	Elbow angular velocity (rad/s) in local coordinates
11	SHO_ACC	Shoulder angular acceleration (rad/s2) in local coordinates
12	ELB_ACC	Elbow angular acceleration (rad/s2) in local coordinates
13	SHO_TOR_CMD	Commanded shoulder torque (Nm) in local coordinates
14	ELB_TOR_CMD	Commanded elbow torque (Nm) in local coordinates
15	M1_TOR_CMD	Commanded M1 torque (Nm) in global coordinates (torque on segment L1 produced by motor M1)
16	M2_TOR_CMD	Commanded M2 torque (Nm) in global coordinates (torque on segment L2 produced by motor M2)
17	L1_ANG	L1 angle (rad) in global coordinates
18	L2_ANG	L2 angle (rad) in global coordinates
19	L1_VEL	L1 angular velocity (rad/s) in global coordinates
20	L2_VEL	L2 angular velocity (rad/s) in global coordinates
21	L1_ACC	L1 angular acceleration (rad/s2) in global coordinates
22	L2_ACC	L2 angular acceleration (rad/s2) in global coordinates
23	HPX	Position of calibrated fingertip (m), x component, global coordinates
24	HPY	Position of calibrated fingertip (m), y component, global coordinates
25	HVX	Velocity of calibrated fingertip (m/s), x component, global coordinates
26	HVY	Velocity of calibrated fingertip (m/s), y component, global coordinates
27	HAX	Acceleration of calibrated fingertip (m/s2), x component, global coordinates
28	HAY	Acceleration of calibrated fingertip (m/s2), y component, global coordinates
29	EPX	Position of calibrated elbow (m), x component, global coordinates
30	EPY	Position of calibrated elbow (m), y component, global coordinates
31	EVX	Velocity of calibrated elbow (m/s), x component, global coordinates
32	EVY	Velocity of calibrated elbow (m/s), y component, global coordinates
33	EAX	Acceleration of calibrated elbow (m/s2), x component, global coordinates
34	EAY	Acceleration of calibrated elbow (m/s2), y component, global coordinates

Table 9-1: Arm specific data in KINdata matrix (i.e. KINdata(i,j), where i=1 or i=2)

j	Name	Description
35	SHO_VEL_FLT	Shoulder angular velocity (rad/s) in local coordinates, filtered with 2nd order 10 Hz Butterworth filter
36	ELB_VEL_FLT	Elbow angular velocity (rad/s) in local coordinates, filtered with 2nd order 10 Hz Butterworth filter
37	MOT_STATUS	Motor status bitfield
38	FS_FORCE_U	Force measured by force sensor (N), x component, in local coordinates
39	FS_FORCE_V	Force measured by force sensor (N), y component, in local coordinates
40	FS_FORCE_W	Force measured by force sensor (N), z component, in local coordinates
41	FS_TORQUE_U	Torque measured by force sensor (Nm), x component, in local coordinates
42	FS_TORQUE_V	Torque measured by force sensor (Nm), y component, in local coordinates
43	FS_TORQUE_W	Torque measured by force sensor (Nm), z component, in local coordinates
44	FS_FORCE_X	Force measured by force sensor (N), x component, in global coordinates
45	FS_FORCE_Y	Force measured by force sensor (N), y component, in global coordinates
46	FS_FORCE_Z	Force measured by force sensor (N), z component, in global coordinates
47	FS_TORQUE_X	Torque measured by force sensor (Nm), x component, in global coordinates
48	FS_TORQUE_Y	Torque measured by force sensor (Nm), y component, in global coordinates
49	FS_TORQUE_Z	Torque measured by force sensor (Nm), z component, in global coordinates

Note:All angular positions, velocities and accelerations are defined around the proximal joint for each segment. Segment definitions can be found in Section 9.3 - "KINARM Segment Definitions".

Within the KINdata matrix, row 3 contains data that is not arm-specific, as shown in **Table 9-2**.

Table 9-2: Non-arm-specific data in KINdata matrix (i.e. KINdata(3,j))

j	Name	Description
1	Dominant_arm	Choice of which arm is the ‘dominant’ arm for this task (1 = 1st arm, 2 = 2nd arm)
2	Feed_forward	Duration (s) of feed-forward estimate used for display of hand feedback (i.e. fingertip dot)
3	FDBK_STTS	Feedback status (0=none, 1=dominant, 2=non-dominant, 3=both, 4=controlled by Task Program).
4	FDBK_SRC	Source of feedback style, color etc. (0=from BKIN Dexterit-E GUI, 1=from Task Program)

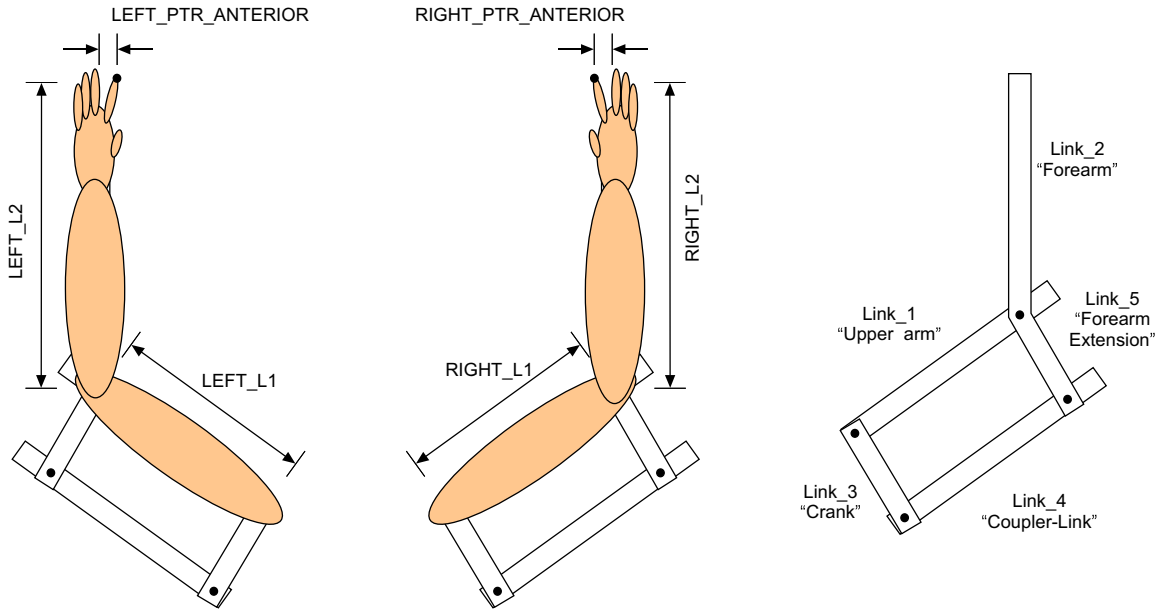
Table 9-2: Non-arm-specific data in KINdata matrix (i.e. KINdata(3,j))

j	Name	Description
5	FDBK_CLR	Feedback color (RRRGGGBBB)
6	FDBK_SIZE	Radius of feedback dot (m)
7	ENBL_LOADS	Status of load enable from BKIN Dexterit-E
8	VEL_DLY_PRI	Delay of velocity signals (s) from primary encoders. Delay is produced by motion control card on-board calculations.
9	ACC_DLY_PRI	Delay of acceleration signals (s) from primary encoders. Delay is produced by motion control card on-board calculations.
10	VEL_DLY_SEC	Delay of velocity signals (s) from optional secondary encoders.
11	ACC_DLY_SEC	Delay of acceleration signals (s) from optional secondary encoders.

9.3 KINARM Segment Definitions

The various segments of the KINARM robot, as well as the subject arm lengths that are calibrated during subject setup with a KINARM Exoskeleton robot, are shown from a top-down view in **Figure 9-1**. The segments of the KINARM End-Point robot follow the same numbering scheme.

Fig 9-1. KINARM Segment Definitions



9.4 Stateflow Events versus Event Codes

Stateflow events should not be confused with user-defined event codes. Stateflow events are Stateflow objects that drive Stateflow diagram execution. They are not saved in a data file for subsequent data

analysis. Event codes are user-defined codes that have no effect on a Stateflow chart, but they are saved in a data file for subsequent data analysis.

9.5 Global Coordinate System

All data accessed in a Task Program is defined in the same, single, global coordinate system. The global coordinate system is defined with positive X to the right and positive Y towards the front (away from the subject). The origin (0,0) is defined as the middle of the bottom of the computer generated image shown to the subject. These data include the KINARM kinematic data (see Section 9.2 - "Structure of KINdata") and all video data (e.g. VCODEs and the Target Table). Thus there is a difference between the values in the Target Table viewed by a user in BKIN Dexterit-E and the values in the Target Table as accessed by a Task Program. In BKIN Dexterit-E, the user sees and defines the Target Table positions in a local coordinate system (e.g. relative to the fingertip when the arm is in a certain orientation). When a Task Protocol is chosen and downloaded, however, those positions get translated to the global coordinate system for the Task Program, so that they are in the same coordinate system as everything else. This translation is automatic. Most of these data are defined in SI units, with the exception of the Target Table which presently uses cm.

Although the origin is always at the middle of the bottom of the computer generated image, this does not always correspond to the middle of the bottom of the image viewed by the subject. Two factors can lead to the global origin not being at the middle of the bottom of the viewable image: incomplete image projection in a back-projection system and/or a mismatch in resolutions between the display device and Windows. Neither of these problems typically occurs with LCD monitors, whereas both often occur with project-based back-projection systems.

In a back-projection image system, the problem of incomplete image projection occurs if the screen is not large enough to display the entire projected image. If this situation occurs, then part of the projected image will be cut off and the origin may appear off-screen.

The other potential problem with projectors occurs when there is a mismatch in resolutions between the display device and Windows. More specifically, if the projector's aspect ratio setting is set to 'Native' and there is a mismatch between the projector's native resolution and the resolution chosen in the Windows Display Properties then part of the image will be cut off and the origin will not be at the middle of the bottom of the projected image. For example, consider a wide-screen projector with a native 1280 x 848 resolution and an aspect ratio setting set to 'Native'. Under these conditions, the projector will only display the first 1280 x 848 pixels that it receives. If the screen resolution in the Windows Display Properties is set to 1280 x 1024, which is larger than 1280 x 848, then part of the computer generated image will not be displayed. More specifically, only the first 848 lines of the computer generated image will be displayed. The remaining 176 lines (i.e., 1024 - 848 = 176) will not be displayed. So in this example the origin, which is part of the undisplayed image, will not be projected.

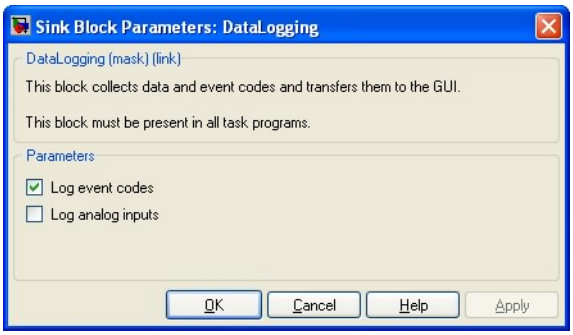
9.6 Data Saving and Logging

Data collected by a Task Program are saved as individual trials in the open source C3D format. For more information on this file format and how to access the data, please see the Dexterit-E User Guide.

Which data are logged by a given Task Program depends on the `DataLogging` block. If the Log event codes check box is checked, then all event codes input to the `event_codes` input of the `DataLogging` block are recorded. If the Log analog data check box is checked, then any signals that are input into the `analog_data` input are recorded.

KINARM kinematic data (e.g. positions, velocities, torques, etc.) are recorded automatically.

Fig 9-2. Screenshot of DataLogging Dialog



When data are logged depends on the logging_enable input and on the run/pause state of a task. Data are only saved if `logging_enable=1` and if the task is running. Data are not saved when a task is paused.

9.7 Available ‘Tags’ (From and Goto Blocks)

Example 6.16 - "Accessing KINdata, Current Block Index, and Other Task Control Variables" demonstrated how to use Simulink’s From block to access Goto tags. A list of available tags and what data they contain is provided here.

`Block_Definitions` – Partial copy of the Block Table for the current Task Protocol, albeit in a different format than seen by users in BKIN Dexterit-E. In this format, the columns of `Block_Definitions` are as follows:

1 – Randomize (0 = non-random, 1 = random);

2 to 500 – list of Trial Protocols.

`Block_Sequence` – Partial copy of the Block Table for the current Task Protocol, albeit in a different format than seen by user in BKIN Dexterit-E. In this format, the columns of `Block_Definitions` are as follows:

1 – Block index (i.e. row number of Block Table or `Block_Definitions`)

2 – Number of repeats of block defined by block index.

`KINARMTorqueCMD` – A 1x4 vector of current torque command to KINARM robots. See Help for the `KINARM_Apply_Loads` block for description of elements. This tag is only available if the `KINARM_Apply_Loads` block is present in the model.

`KINARM_docking_points` – Location (x,y) of KINARM exoskeleton’s calibration docking points in the global coordinate system (m).

`KINdata` – Matrix containing data relevant to current KINARM Lab.

`Load_Table` – Complete copy of the Load Table for the current Task Protocol.

`Run_Status` – Status of task.(0 = stopped/paused, 1 = running)

`TP_Table` – Complete copy of Trial Protocol table.

`Target_Table` – Complete copy of Target Table. When the target table is accessed in a Task Program, the positions of targets are in global coordinates, not in the local coordinates viewed in BKIN Dexterit-E.

`current_block_index` – Block index of current block. The block index refers to the row in the Block Table in BKIN Dexterit-E (or the row of the `Block_Definitions` constant in the TDK).

`current_block_number_in_set` – Block number of current block. In the Block Table in BKIN Dexterit-E, a single row may define multiple instances of identical blocks (i.e. if Block Reps > 1). Each of those repeats counts as a separate block for the purposes of this counter. This number begins at 1 and increments with each new in block throughout the Task. Its final value will equal the total number of blocks defined in the Block Table (i.e. sum of values in the Block Reps column).

`current_tp_index` – Row number of current Trial Protocol in the `TP_Table` (i.e. can be used with `TP_Table` to extract the current trial protocol).

`current_trial_number_in_block` – Number of current trial in block. This number begins at 1 for each block and increments with each new trial throughout that block. If a Task is paused between blocks, then current_trial = 0 will occur during that pause.

`current_trial_number_in_set` – Number of current trial in set. This number begins at 1 for each set and increments with each new trial throughout the set.

`display_size_m` – Size of the subject display (x, y) in meters.

`display_size_pels` – Size of the subject display (x, y) in pixels.

`global_clock` – Square wave signal at 1 kHz.

`last_frame_acknowledged` – Frame number of the last video frame that was acknowledged as displayed by the GUI computer.

`last_frame_sent` – Frame number of the last video frame that was sent to the GUI computer.

`primary_encoder_data` – Matrix of robot data determined from the primary encoders. This tag allows access to primary encoder data even when using optional secondary encoders (KINARM EP only). This matrix has two rows, one for each robot. Each row has six elements:

1 – L1 position (rad)

2 – L2 position (rad)

3 – L1 velocity (rad/s)

4 – L2 velocity (rad/s)

5 – L1 acceleration (rad/s^2)

6 – L1 acceleration (rad/s^2)

`robot1_calibration_parameters` – Vector of information about the calibration of robot 1. Elements of this vector are:

1 – Shoulder angle offset (rad)

2 – Elbow angle offset (rad)

3 – Shoulder x position (m)

4 – Shoulder y position (m)

5 – L1 length (m)

6 – L2 length (m)

7 – Offset to pointer from L2 (m)

8 – L3 error estimate (m)

`robot2_calibration_parameters` – Vector of information about the calibration of robot 2, if present. The format of this vector is the same as `robot1_calibration_parameters`.

`robot1_motor_parameters` – Vector of information about the hardware of robot 1. Elements of this vector are:

1 – orientation (1 = right exoskeleton / left endpoint, -1 = left exoskeleton / right endpoint)

2 – motor 1 orientation (1 = motor shaft down, -1 = motor shaft up)

3 – motor 2 orientation (1 = motor shaft down, -1 = motor shaft up)

4 – motor 1 gear ratio

5 – motor 2 gear ratio

6 – using secondary encoders (1 = using secondary encoders, 0 otherwise; KINARM EP only)

7 – robot type (1 = endpoint, 0 = exoskeleton)

8 – torque constant

9 – using force transducers (1 = using force transducers, 0 otherwise; KINARM EP only)

`robot2_motor_parameters` – Vector of information about the hardware of robot 2, if present. The format of this vector is the same as `robot1_motor_parameters`.

`run_state` – State of Task Program running on the real-time computer, with the following definitions:

1 – Task is ready but has not yet started

2 – Task is running

3 – Task is paused

4 – Task is finished

`subject_height` – Subject height as entered into the Dexterit-E user interface (m).

`subject_weight` – Subject weight as entered into the Dexterit-E user interface (kg).

Deprecated Goto Tags in TDK 2.3

The following Goto tags are no longer available in TDK 2.3:

`Block_Table` – The functionality of this Goto tag has been replaced with the `Block_Definitions` and `Block_Sequence` Goto tags.

`current_block` – This Goto tag has been renamed `current_block_number_in_set`.

`current_megablock` – This Goto tag has been renamed `current_block_index`.

`current_TP_index` – This Goto tag has been renamed `current_tp_index`.

`current_trial` – This Goto tag has been renamed `current_trial_number_in_block`.

`system_event_code` – No longer available.

`trial_queue_length` – No longer available

`trial_queue_list` – The functionality of this Goto tag has been replaced with the `Block_Definitions` and `Block_Sequence` Goto tags.

`TP_row` – This Goto tag has been renamed `current_TP_index`.

9.8 GUI Control Input Events and Output Events

e_clk – This input event must be on port 1 and must have a defined trigger of Rising. This event occurs every 1.0 ms, as long as the Task Program is running (i.e. as long as the task has started and is not paused). This event causes every Stateflow Chart to which it is wired to execute once per ms (because Stateflow Charts only execute when an event occurs). A typical secondary usage of e_clk is as a time (e.g. see usage of `after` in **Figure 3-7**).

e_ExitTrialNow – This input event must be the on port 2 and must have a defined trigger of Either. This event occurs whenever a user clicks Pause and BKIN Dexterit-E has been set to pause the trial immediately. This event can be used by the end user to define what will happen in the Task Program when this event occurs. For an example, see Section 6.3 - "Pause Button Control (and Stateflow Sub-States and Stateflow Events)"

e_Trial_End – This output event can be on any port, but must have a defined trigger of Either. This event is used by the end user to signal when a trial is over so that the Task Program can update itself for the next trial (e.g. see usage in `Between_Trials` state in **Figure 3-7**).

9.9 Updating Task Programs from Prior Versions of the TDK

Differences between TDK versions are documented in the BKIN Dexterit-E release notes. This section explains the changes required in an existing Task Program when updating to a later TDK version.

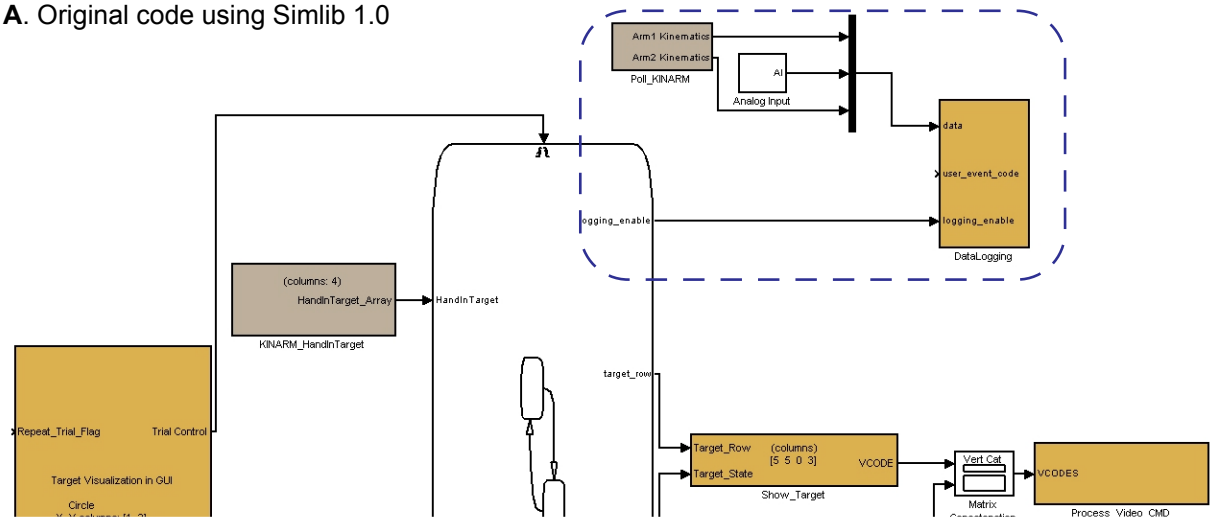
Note:Versions of the task development kit distributed with BKIN Dexterit-E 2.2 and earlier were referred to as the Dexterit-E Simulink Library, or “Simlib”.

Updating Task Programs from Simlib 1.0 to 1.1

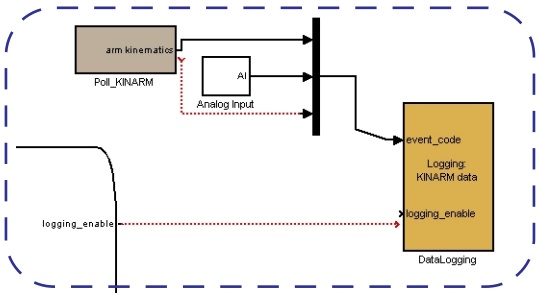
Updating a Task Program from Simlib 1.0 to 1.1 requires a number of steps. The first step to updating a Task Program is to install Simlib 1.1. This is then followed by updating the Matlab path: the old paths to the Simlib 1.0 directory must be removed, the new paths to the Simlib 1.1 directory must be added, and the new paths saved. Matlab should be closed and restarted after updating the path to ensure that the changes take effect properly. Task Programs must then be opened, and the changes described below must be made prior to rebuilding each Task Program.

Fig 9-3. Updating from Simlib 1.0 to 1.1

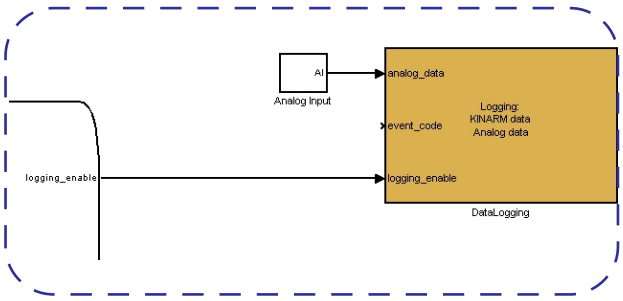
A. Original code using Simlib 1.0



B. After updating the Matlab path to Simlib 1.1



C. Updated code



- Delete `Poll_KINARM`.
- Delete the `mux` block.
- Connect the `AI` output from the `Analog Input` block to the `analog_data` input of the `DataLogging` block.

Updating Task Programs from Simlib 1.1 to 1.2

Task Programs built with Simlib 1.2 no longer need to reference the `sm_common_v2.h` and `sm_common_v2.c` files. These two files can be deleted, but the Task Programs need to be updated to stop referencing them. The following actions will remove the references to these files in a Task Program.

- Within the Task Program's Simulink code, choose Simulation→ Configuration Parameters. In the left panel of the Configuration Parameters dialog, select Real-Time Workshop→ Custom Code, and then select Source files in the bottom middle right panel. Remove `sm_common_v2.c` from the list of source files.

  Still within the Real-Time Workshop→ Custom Code selection in the Configuration Parameters dialog, select Header file in the top middle right panel. Remove `#include "sm_common_v2.h"` from the list of include statements.

- Within the Task Program's Stateflow Code, choose Tools→ Open Simulation Target. In the Simulation Target dialog, select the Custom Code tab. Within the Custom Code tab, select the Source Files tab. Remove `sm_common_v2.c` from the list of source files.

Still within the Custom Code tab of the Simulation Target dialog, select the Include Code tab. Remove `#include "sm_common_v2.h"` from the list of include statements.

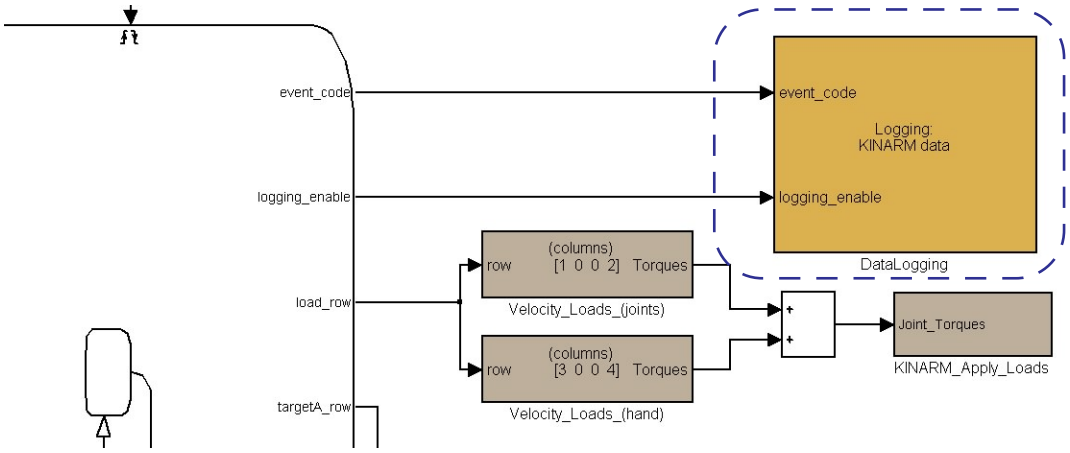
Updating Task Programs from Simlib 1.2 to TDK 2.3

A number of changes have occurred to task development kit library blocks that may produce warnings and/or unexpected behavior.

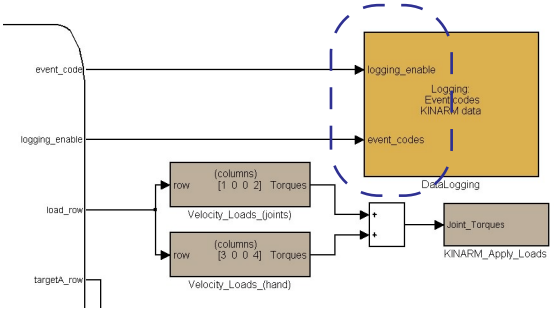
- Upon first opening a Task Program after updating to TDK 2.3, the following warnings may be displayed at the Matlab prompt. These warnings can safely be ignored.
    - Warning: In instantiating linked block 'your_task/GUI Control' : GUI Control block (mask) does not have a parameter named 'SimlibVersion'.
    - Warning: In instantiating linked block 'your_task/DataLogging' : DataLogging block (mask) does not have a parameter named 'log_kinarm_data'.
    - Warning: In instantiating linked block 'your_task/DataLogging' : DataLogging block (mask) does not have a parameter named 'log_analog_data'.
- The names and meanings of various Goto tags have changed. If the Task Program uses any of the previously defined Goto Tags (e.g. Section 6.16 - "Accessing KINdata, Current Block Index, and Other Task Control Variables") refer to Section 9.7 - "Available ‘Tags’ (From and Goto Blocks)", which lists the currently available Goto tags as well as those that have been deprecated in TDK 2.3.
- As shown in **Figure 9-4**, the order of the inputs to the DataLogging block has changed. This may or may not cause errors in wiring, requiring manual rewiring of the inputs.

Fig 9-4. Updating from Simlib 1.2 to TDK 2.3

A. Original code using Simlib 1.2



B. After updating the Matlab path to TDK 2.3



C. Updated code ready to build with TDK 2.3