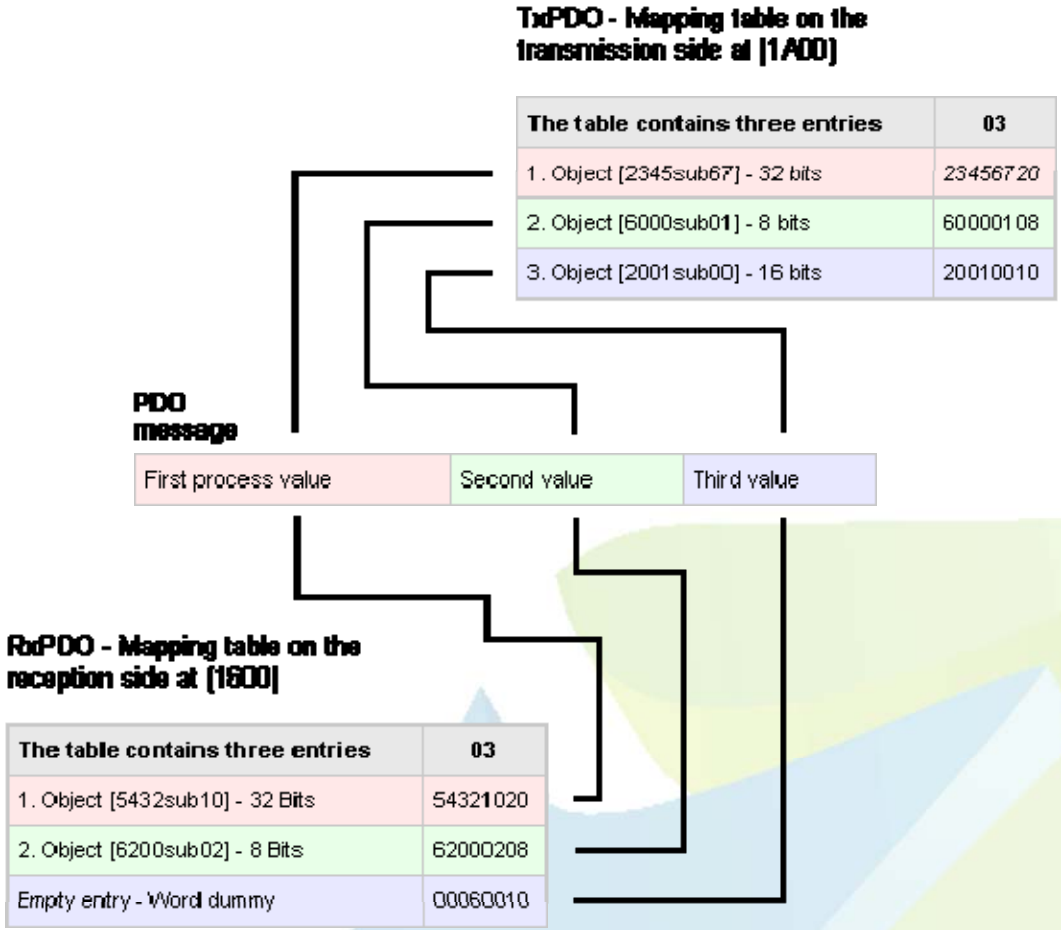
PDO 를 이용한 처리 데이터 교환

CANopen 시스템의 핵심 작업은 당연히 처리 데이터의 교환입니다. 이를 위해 대부분의 CAN 식별자들 뿐만 아니라 객체 사전 엔트리의 대부분이 제공됩니다. 처리 데이터의 전송, 이것은 별도의 프로토콜 오버헤드 내에서 발생하며 전송은 이른바 **"producer-consumer 원칙"**에 따라 이루어집니다. 이는 노드("producer")에 의해 전송된 메시지가 다른 모든 노드들("consumer")에 의해 수신될 수 있음을 의미합니다. 이 원리는 **"방송(broadcasting)"** 으로 일컬어지며 대표적인 데이터 전송의 매우 효율적인 방법으로, 특히 메시지가 (예. synchronization message) 모든 처리 참가자들에게 동시에 전달되어야 하는 경우가 좋은 예입니다.

처리 데이터를 포함하고 있는 CAN 메시지를 PDO (**"Process Data Object"**)라 부릅니다. 이미 설명한대로, PDO 의 전송은 "기능적("Operational") 상태에서만 가능합니다. PDO 는 정해진 형태가 없습니다. PDO 의 데이터 영역은 1 부터 8 data byte 길이 사이에서 가능합니다. 또한 PDO 의 내용은 쉽게 해석될 수 없습니다. 여기서의 기본 착상이 전송자와 수신자 모두가 PDO 의 내용이 어떻게 해석될지 안다는 것입니다. 이런 이유로 PDO 를 오직 이것의 COB-ID 에 의해서 식별하는 것이 효율적입니다. 소위 "**"PDO mapping"** 은 PDO 의 데이터 필드에 있는 어떠한 개별 처리 변수들이 전송되는지, 그들이 어떻게 배열되는지 그리고 어떤 데이터 유형과 길이를 갖는지를 기술합니다. 따라서 각각의 정의된 PDO 의 데이터 영역의 내용과 의도는 전송측과 수신측 모두의 객체 사전 내에서 PDO-mapping record 의 형태로 서술되어 있습니다.

PDO producer 는 이것의 TxPDO mapping 대로 PDO 의 데이터 영역을 구성합니다. 이를 위해 객체 사전에서 전송되는 변수들의 최신 데이터를 가져와서 CAN 메시지 (PDO)가 전송되기 전에 PDO 의 데이터 영역에 이것들을 복사합니다. 동일한 일들이 consumer 측에서도 발생합니다. RxPDO mapping 기록을 바탕으로, 수신된 PDO 의 데이터 바이트들이 로컬 객체 사전 엔트리로 복사되고, 따라서 일반적으로 장치-특정 행동들이 트리거됩니다 (예. 디지털 출력 설정).
특정한 PDO 를 수신하기 원하는 네트워크 노드는 해당 CAN 메시지를 반드시 활성화해야 합니다. 이것은 OD 에 있는 관련 COB-ID 엔트리의 "set valid"로 이루어집니다.

이제 다시 PDO mapping 으로 돌아가서. 처리 변수의 배열(mapping) 원칙은 아래 그림과 같습니다 (변수들은 애플리케이션 프로파일에서 객체 사전 엔트리 형태로 이용할 수 있습니다). PDO 데이터 영역에 있는 개별 처리 변수들의 매핑은 표 형식으로 설명되어 있습니다. 또한 이것은 객체 사전 엔트리로, 즉 [16xx] 또는 [1Axx] 의 모든 전송과 수신-PDO 에 관한 것으로 제공됩니다. 그러므로 이러한 표들은, PDO 의 데이터 영역에서 SDO write 액세스를 통해 처리 변수의 매핑을 구성할 수 있습니다.



이 예에서는 정확하게 두 개의 오브젝트 연결들이 있습니다: PDO producer 의 오브젝트 (처리 변수) [2345sub67] 부터 PDO consumer 의 오브젝트 [5432sub10] 까지 그리고 producer 의 오브젝트 [6000sub01]부터 consumer 의 오브젝트 [6200sub02]까지. 세번째 전송 오브젝트, [2001sub00] 는 수신측에서 평가되지 않기 때문에 이른바 모조 오브젝트(dummy object)로 덮어집니다.

매핑 표는 subindex 0 을 포함하고 있는 "PDO mapping" 데이터 유형에서 오는데, subindex 0 에는 매핑된 오브젝트의 숫자 그리고 subindex 1 부터 8 까지에는 DWORD 로서 매핑된 객체 사전 엔트리 자체가 포함됩니다. 이것은 24 비트 길이의 OD 주소 (index, subindex)와 8 비트 부호 길이의 OD 엔트리를 포함합니다. 8 개의 매핑 엔트리들을 지원하는 장치는 8 의 세분성(granularity)을 가지며 따라서 "byte-wise mapping" 을 실행할 수 있습니다. 한편 1 의 세분성을 가진 장치는 각각 개별적인 PDO bit 의 매핑을 지원합니다 ("bit-mapping"). 이에 따라 매핑 표에는 64 개의 엔트리들이 반드시 있어야 합니다.

그러나 PDO 는 매핑(mapping) 뿐만이 아니라 통신 파라매터들에 의해서도 설명됩니다.
여기에는 다음과 같은 것들이 있습니다: COB-ID, 즉 CAN 메시지의 식별자, "전송 유형", "inhibit time", 그리고 관련 객체 사전 엔트리에 의해 구성될 수 있는 모든 것. 이러한 추가 객체 사전 엔트리의 위치는 연관된 매핑 엔트리, 즉, 전송 PDO [18xx] 와 수신 PDO [14xx] 보다 이전의

0x200 오프셋을 갖는다. 따라서 각각의 PDO 는 두 개의 서로 다른 OD 엔트리에 의해 기술되는데, 이것들은 확실하게 함께 있어야 하며 두 개 모두 시스템 통합자(system integrator)에 의해 구성되어야 합니다. 아래의 표는 "PDO parameter" 기록입니다.

Sub-Index	Contents	Data type
0	Largest sub-index supported	BYTE
1	COB-ID	DWORD
2	Type	BYTE
3	Inhibit time in ms	WORD
5	Event timer	WORD

Subindex 1 은 오른쪽-정렬의 더블 워드(double word)로 된 PDO 의 CAN 식별자를 포함합니다. 가장 큰 값의 비트는 PDO 가 active (유효) 또는 inactive (무효)인지를 나타냅니다. MSB 세트는 무효를 뜻하며, 0x80000199 의 값은, 예를 들면, 25 번의 노드의 무효한 첫번째 TxPDO 를 나타냅니다.

PDO 의 전송 유형은 두번째 subindex 를 통해 설정할 수 있습니다. 먼저 동기적 PDO 와 비동기적 PDO 를 구분하는 것이 필요합니다.

Value (dec)	Type
0	acyclic synchronous
1...240	cyclic synchronous
241...251	reserved
252	synchronous RTR only
253	asynchronous RTR only
254...255	asynchronous

비동기적 PDO 는 사건-제어(event-controlled) 방식으로 PDO 의 일반적인 전송 유형을 나타냅니다. 이것은 PDO 에 매핑되어 있는 처리 변수들 중에서 적어도 하나, 가령 입력 값이 변경될 때, PDO 가 즉각 전달되는 것을 뜻합니다. 이를 위해 255 또는 254 값들은 PDO 타입으로 기입됩니다.

동기적 PDO(***Synchronous PDO)*** 는 동기적 메시지(Sync Object) 를 우선 수신한 후에 전송됩니다. 따라서 PDO 전송은, 거의 동시에, 전체 네트워크에서 동기적으로 실행됩니다.
그러나 이보다 중요한 것은 모든 디바이스 인풋들이 sync 오브젝트 도착시 샘플이 되어야만, 처리 과정의 균일한 스냅샷이 도출된다는 것입니다. 그 다음 sync-message 와 함께, 기록된 데이터가 동기적 PDO 에서 전송됩니다. 그러므로 여기서는 Sync 메시지의 주기 시간(cycle time)에 해당하는 지연이 있게 되는데, consumer 가 이전의 Sync message 당시의 처리 변수들을

수신하기 때문입니다. 아웃풋 쪽에서는 노드에 의해 수신된 동기적 PDO 들이 그 다음 Sync message 의 도착시 유효하게 됩니다.

각각의 Sync 메시지와 같이 모두 전송되는, 수많은 동기적 PDO 에 의해 버스가 막히지 않게 하기 위해, 전송 간격의 분할기(divider)로 주기적인 동기적 PDO 타입의 1..240 수치가 사용됩니다. 이에 따라 [18xxsub02] = 4 는 동기적 PDO 가 각각의 네번째 Sync 메시지와 함께 전송된다는 것을 뜻합니다. 이것에 영향 받지 않는, 디바이스는 당연히 모든 Sync 메시지와 함께 그 인풋을 기록해야 하는데, PDO 에서 동기적으로 기록된 처리 변수들이 RTR 에 의해 요구될 수 있기 때문입니다. 이 경우 CANopen 디바이스는 동기적으로 기록된 값들과 함께 즉시 관련 PDO 를 전송해야 합니다. [1006], "Communication Cycle Period" 파라미터에서, 노드들은 마이크로초(microsecond)의 Sync 간격으로 보고될 수 있습니다.

끝으로, subindex 3 의 **"inhibit time"**을 비동기적 PDO 를 위해 설정할 수 있습니다. 그 값은 100 microseconds 의 배수로 주어지게 됩니다. inhibit time 은 계획성 없는 인풋 값의 빈번한 변경이 발생할 때 유용합니다, 가령 open analog input. 만약 inhibit time 이 구성되면 노드는 inhibit time 의 종료 전에는 관련 PDO 를 다시 전송하지 않을 것입니다; 이는 소위 **"babbling idiot"**로 인한 허용 불가능한 높은 busload 가 없음을 보증합니다. inhibit time 은 당연히 TxPDO 에만 사용됩니다. 이것은 RxPDO 에서는 전혀 의미가 없습니다.

비동기적 TxPDO 는 **event timer**, subindex 5 와 함께 주기적으로 전송될 수 있습니다. 이 값이 10 보다 크다면, 이는 millisecond timer 가 됩니다. 이것이 종료될 때 PDO 가 전송됩니다. 그러므로 전송은 외부 디바이스 인풋이 바뀌고 event timer 가 끝날 때 발생합니다. 이 subindex 또한 transmit-PDO 에서만 의미가 있을 뿐입니다.

CANopen 노드는 일반적으로 4 개의 전송-POD 들과 4 개의 수신-PDO 들을 갖고 있으며, 하나의 COB-ID 가 이러한 PDO 들 각각을 위해 준비되어 있습니다. 따라서 이것은 총 127*4*2 = 938 COB-ID 들을 차지합니다. 즉 전체 식별자(identifier) 공간의 거의 절반입니다. 그러나 이른바 "**object linking**", 다시 말하면 전송과 수신-PDO 사이의 통신 관계 수립으로 인하여, CAN 식별자들은 다시 자유로워지는데, producer 또는 consumer 연결은 이것의 COB-ID 를 통신 파트너의 것에 맞추어야하기 때문에 예비된 자신의 COB-ID 는 따라서 다시 자유로워집니다. 그러므로, 실제로, 127 개의 네트워크 노드가 있는 네트워크의 경우, 디바이스 한 개당 평균 8 개의 COB-ID 를 TxPDO 에 이용할 수 있습니다. 이보다 낮은 네트워크 노드들이 있는 네트워크는, 당연히 미사용 COB-ID 들도 사용될 수 있습니다. 시스템 통합자는 항상 실제로 사용되는 COB-ID 와 식별자 공간의 할당에 관한 전체적인 시야를 갖고 있어야 합니다; 더욱 복잡한 시스템들을 위해서, 소프트웨어 툴 – 가령, IXXAT CANopen ConfigStudio 등이 권장됩니다. 예를 들어 이 툴은, PDO-mapping 과 –linking 를 자동으로 처리합니다.