Application Note 001:

"Conveyor control using the S1000 Smart RTU and Web Services"

1 Introduction

This application note describes how to implement a simple conveyor control, first as a stand-alone application, and later adding Web Services for monitoring and control.

1.1 Conveyor structure

The main elements of the conveyor to be used in this application note are illustrated in Figure 1. A conveyor belt is actuated by a motor. Products (illustrated by a rectangular box) are conveyed from left to right. A proximity sensor (red beam) detects the presence of a product at around the centre of the conveyor. An RFID reader can be used to acquire an identification code for the product. Finally, a push button is used by an operator to control the station.

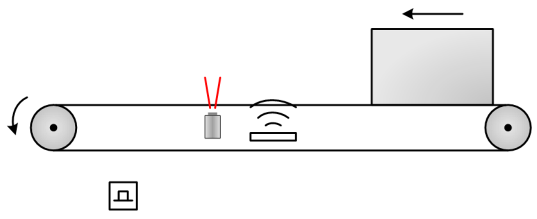


Figure 1. Conveyor diagram.

1.2 RFID identification

The RFID reader is connected to the S1000 via the RS232 serial port. An RFID code is read by sending the character sequence 'R' + CR + LF. The RFID reader returns a 32-bit code followed by the CR + LF characters.

1.3 Web Services

The Inico S1000 offers a built-in Web Services engine, which enables the device to exchange data with any modern PC application. The engine conforms to the Devices Profile for Web Services[1], which specifies the functions that are typically required by embedded systems. The S1000 sends and receives messages in standard XML and SOAP, and therefore can very easily communicate with applications written in C, C++, C#, Java, Visual Basic, and many other languages.

1.3.1 Events

Event messages are XML/SOAP messages which are produced by the S1000 when particular conditions are met. In order to receive a copy of an event message, an application must first *subscribe* to the S1000. The S1000 can then send multiple copies of the event message to all registered subscribers.

[1] http://schemas.xmlsoap.org/ws/2006/02/devprof/

The conveyor application will generate two types of event messages:

- *ItemTransferIn* events: generated when a product is transferred in by the conveyor.
- *ItemTransferOut* events: generated when a product is transferred out by the conveyor.

1.3.2 Commands

Command messages are XML/SOAP messages which carry an instruction to perform an operation. A response may be sent back acknowledging the request, or refusing it if the operation is not possible or not understood. An empty response is by convention understood to mean that the command was accepted.

The conveyor application shall accept a single type of command message:

- *TransferInCommand* message: instructs the conveyor to start moving in order to receive a product.

2 Simple control implementation

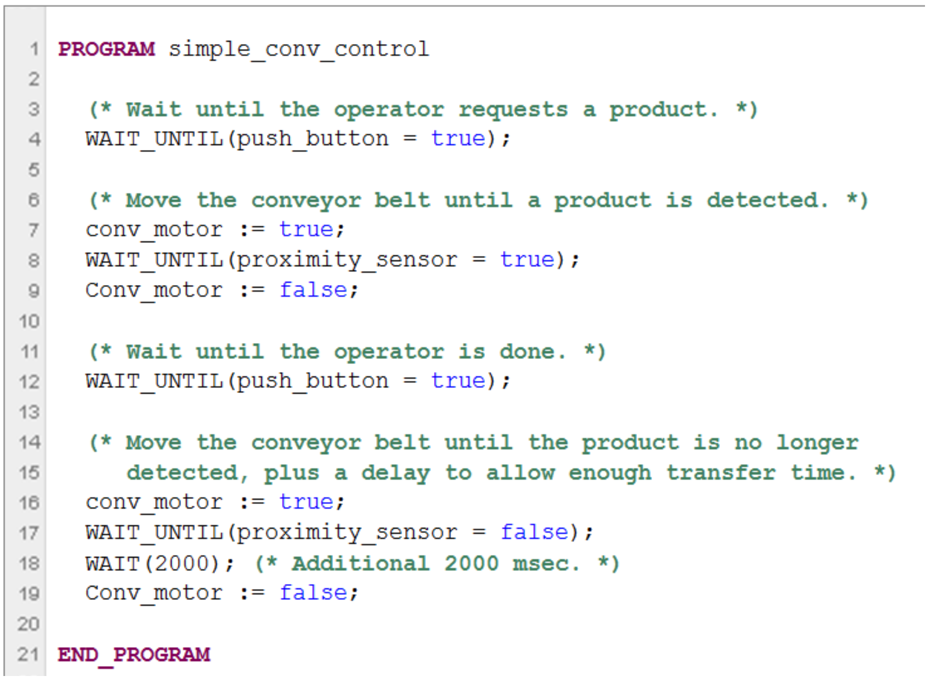
2.1 Operator-controlled material flow

The first version of the conveyor control application will be controlled by the operator through the available pushbutton, and shall not use Web Services yet.

A typical product cycle can be described by the following sequence:

1. The operator activates the pushbutton, causing the conveyor belt to start moving.
2. A product is transferred in. When the proximity sensor detects it in position, the conveyor belt is stopped.
3. The operator performs an operation on the product, such as mounting a part, placing user manuals in a box, or inspecting quality.
4. When the operation is complete, the operator activates the pushbutton, and the conveyor belt transfer the product to the next step in the production line.

The following listing shows the ST program required to perform the aforementioned steps.



```
PROGRAM simple_conv_control

  (* Wait until the operator requests a product. *)
  WAIT_UNTIL(push_button = true);

  (* Move the conveyor belt until a product is detected. *)
  conv_motor := true;
  WAIT_UNTIL(proximity_sensor = true);
  Conv_motor := false;

  (* Wait until the operator is done. *)
  WAIT_UNTIL(push_button = true);

  (* Move the conveyor belt until the product is no longer
     detected, plus a delay to allow enough transfer time. *)
  conv_motor := true;
  WAIT_UNTIL(proximity_sensor = false);
  WAIT(2000); (* Additional 2000 msec. *)
  Conv_motor := false;

END_PROGRAM
```

2.2 RFID identification of product

In this section, the conveyor control program is furnished with product identification code. It assumes that the following helper variables have been declared: c : INT; itemCode : INT; itemCodeHex : STRING(16). (Parts of the code shown in the previous section have been removed for clarity, for the complete listing see Appendix A).

```
PROGRAM simple_conv_control
  (* ... transfer product in ... *)

  (* Send read command to RFID reader on COM port 0. *)
  SERIAL_WRITE(0, 'R$R$N');

  (* Wait 100msec and read response: 32 bits, 8 bits at a time. *)
  WAIT(100);
  c := SERIAL_GETC(0);
  itemCode := c;
  c := SERIAL_GETC(0);
  itemCode := itemCode * 256 + c;
  c := SERIAL_GETC(0);
  itemCode := itemCode * 256 + c;
  c := SERIAL_GETC(0);
  itemCode := itemCode * 256 + c;
  INT_TO_HEX(itemCode, itemCodeHex); (* Convert to HEX string, for an
                                        easy to read representation. *)
  c := SERIAL_GETC(0); (* discard carriage return. *)
  c := SERIAL_GETC(0); (* discard line feed. *)

  (* Wait until the operator is done. *)
  WAIT_UNTIL(push_button = true);

  (* ... transfer product out ... *)
  itemCode := 0; (* Clear product code. *)
  itemCodeHex := ''; (* Clear product code. *)

END_PROGRAM
```

3 SOAP Event messages

In this section, we extend the conveyor control program so that it notifies subscribers (such as a monitoring application, or a product tracking application) when products are transferred in and out of the conveyor.

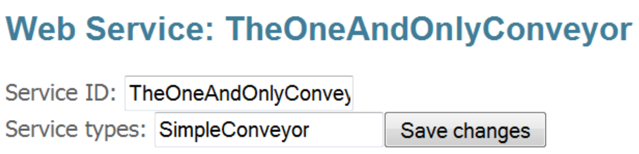
3.1 Web Service

3.1.1 Service configuration

The first step is to create a new Web Service, which will group the configuration for all the messages that are sent and received by the conveyor.

After creating an empty Web Service, two parameters should be configured:

- Service ID: a unique identifier for this service (useful to identify this particular conveyor service). We will use the ID “TheOneAndOnlyConveyor”, but any value is OK.
- Service Types: used to classify the service. We can use “SimpleConveyor” for now.

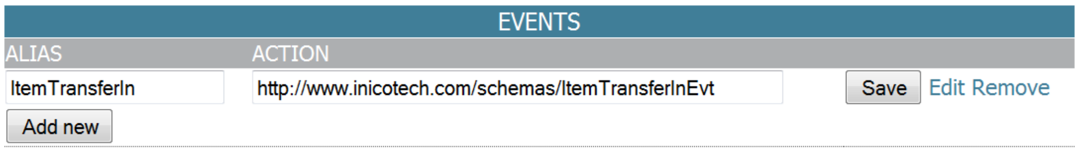


Service ID: TheOneAndOnlyConvey
Service types: SimpleConveyor

3.1.2 *ItemTransferIn event*

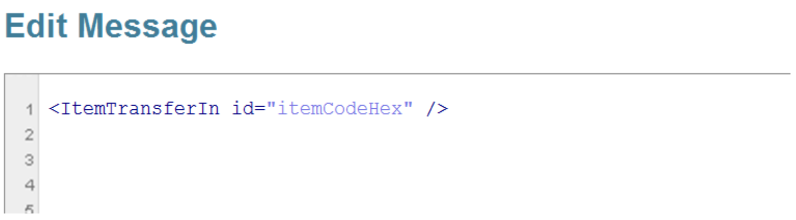
A blank event needs to be added to "TheOneAndOnlyConveyor" service, which will become the ItemTransferIn event message. Once created, the event is initially configured with the following data:

- Alias: the identifier to be used within ST logic for this message. Use "ItemTransferIn".
- Action: identifies this particular type of message, using the WS-Addressing Action field. Use a value such as "http://www.inicotech.com/schemas/ItemTransferInEvt"



ALIAS	ACTION
ItemTransferIn	http://www.inicotech.com/schemas/ItemTransferInEvt

The only thing that now remains is to specify the XML structure for the body of the SOAP message. Using the Edit Body option, the following structure can be used. Note the use of the itemCodeHex variable as an attribute value. When a variable alias is used for the value of an XML parameter or XML element, it is replaced by the actual value of the variable at runtime.

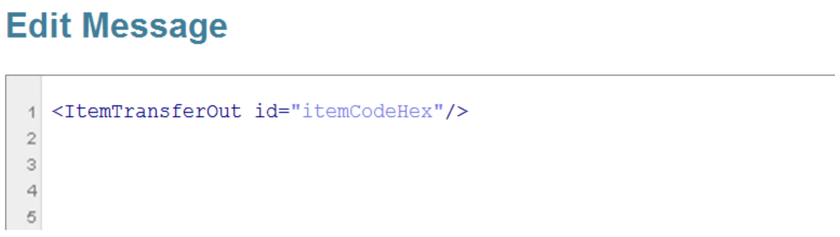


```
<ItemTransferIn id="itemCodeHex" />
```

3.1.3 *ItemTransferOut event*

Add an additional event message, using the following configuration data:

- Alias: "ItemTrasnferOut"
- Action: "http://www.inicotech.com/schemas/ItemTransferOutEvt"
- Body:



```
<ItemTransferOut id="itemCodeHex"/>
```

3.2 ST logic code

Once that event messages have been added to a Web Service, they can be easily published from within an ST logic program. It is important to note that the variables used in the message structure must be set to the appropriate values before the message is published.

```
PROGRAM simple_conv_control

  (* ... transfer product in and read RFID... *)
  INT_TO_HEX(itemCode, itemCodeHex); (* Convert to HEX string, for an
                                         easy to read representation. *)
  WS_PUBLISH(ItemTransferIn);

  (* Wait until the operator is done. *)
  WAIT_UNTIL(push_button = true);

  (* ... transfer product out ... *)
  WS_PUBLISH(ItemTransferOut);

END_PROGRAM

```

4 SOAP Commands

In this section, the conveyor application is further extended to accept a command message.

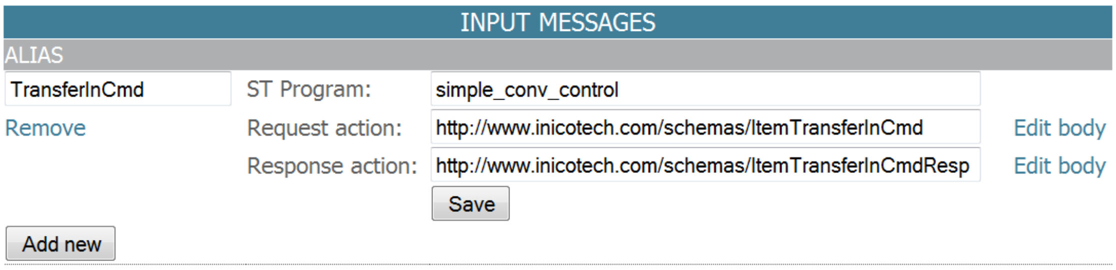
Previously, the conveyor was initially activated for the operator in order to receive a product. This approach requires that the operator be alert to incoming products down the production line. An alternative is to run the conveyor belt continuously until a product arrives. However, this approach is highly inefficient in energy consumption.

The approach adopted here is to wait for a SOAP message to arrive in order to receive a product. This approach assumes that a software application is tracking the movement of products through the factory, and can therefore alert the conveyor when it needs to start moving its belt.

4.1 TransferInCmd message

The first step is to add a blank input message to “TheOneAndOnlyConveyor” service. The message is configured with the following parameters:

- Alias: the identifier to be used within ST logic for this message. Use “TransferInCmd”.
- ST Program: identifies the ST program that will be triggered when the message is received. Use the “simple_conv_control” program.
- Action: “http://www.inicotech.com/schemas/ItemTransferInCmd”
- Response action: “http://www.inicotech.com/schemas/ItemTransferInCmdResp”



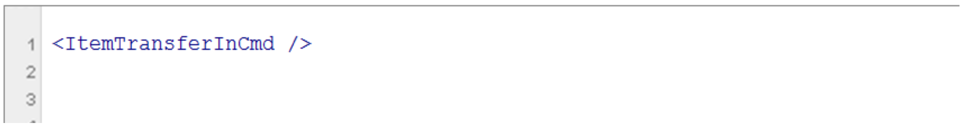
INPUT MESSAGES

ALIAS

TransferInCmd	ST Program:	simple_conv_control	
Remove	Request action:	http://www.inicotech.com/schemas/ItemTransferInCmd	Edit body
	Response action:	http://www.inicotech.com/schemas/ItemTransferInCmdResp	Edit body

- Body:

Edit Message



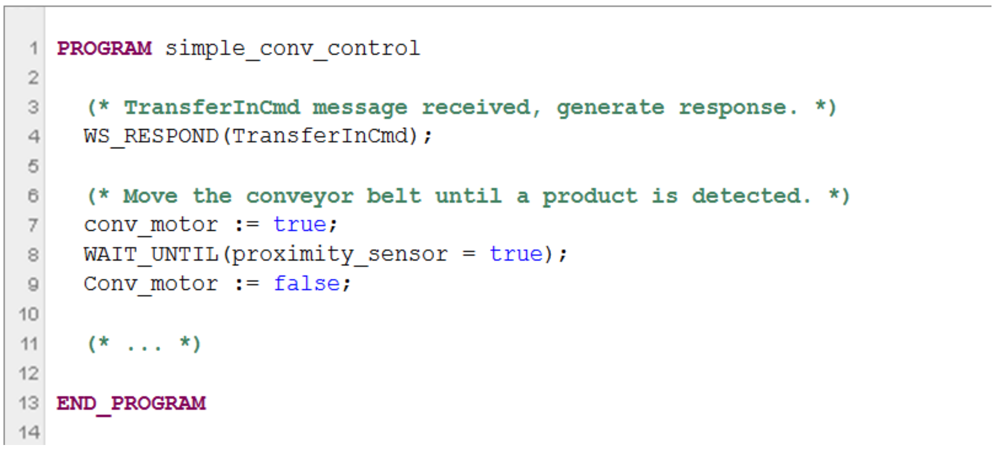
```
<ItemTransferInCmd />
```

- Response body: (leave blank)

4.2 ST logic

An ST program that is associated to an input/command message is no longer executed cyclically. Instead, it is executed once every time that the message is receive. In this case, the “simple_con_control” program is executed once each time a TransferInCmd message is received.

Typically, the first action performed by a program that is triggered by a message is to generate a response. In this case, an empty response is generated, which shall be interpreted as an acknowledgement of the command.



```
PROGRAM simple_conv_control

  (* TransferInCmd message received, generate response. *)
  WS_RESPOND(TransferInCmd);

  (* Move the conveyor belt until a product is detected. *)
  conv_motor := true;
  WAIT_UNTIL(proximity_sensor = true);
  Conv_motor := false;

  (* ... *)

END_PROGRAM
```

Appendix A – Full code listing

```
PROGRAM simple_conv_control

  (* TransferInCmd message received, generate response. *)
  WS_RESPOND(TransferInCmd);

  (* Move the conveyor belt until a product is detected. *)
  conv_motor := true;
  WAIT_UNTIL(proximity_sensor = true);
  Conv_motor := false;

  (* Send read command to RFID reader on COM port 0. *)
  SERIAL_WRITE(0, 'R$R$N');

  (* Wait 100msec and read response: 32 bits, 8 bits at a time. *)
  WAIT(100);
  c := SERIAL_GETC(0);
  itemCode := c;
  c := SERIAL_GETC(0);
  itemCode := itemCode * 256 + c;
  c := SERIAL_GETC(0);
  itemCode := itemCode * 256 + c;
  c := SERIAL_GETC(0);
  itemCode := itemCode * 256 + c;
  INT_TO_HEX(itemCode, itemCodeHex); (* Convert to HEX string, for an
                                        easy to read representation. *)
  c := SERIAL_GETC(0); (* discard carriage return. *)
  c := SERIAL_GETC(0); (* discard line feed. *)
  WS_PUBLISH(ItemTransferIn);

  (* Wait until the operator is done. *)
  WAIT_UNTIL(push_button = true);

  (* Move the conveyor belt until the product is no longer
     detected, plus a delay to allow enough transfer time. *)
  conv_motor := true;
  WAIT_UNTIL(proximity_sensor = false);
  WS_PUBLISH(ItemTransferOut);

  WAIT(2000); (* Additional 2000 msec. *)
  conv_motor := false;
  itemCode := 0;
  itemCodeHex := '';

END_PROGRAM
```