# Not a Batch Language; A Control Language!

Presented at the 1995 World Batch Forum (WBF) in Newtown Square, PA


E. H. Bristol
The Foxboro Co., retired
Foxboro MA, 02035
Tel. (508) 543-8829
email: ebristol@mediaone.net

**KEYWORDS**
batch, control, language, Idioms, Objects, SuperVariable

## ABSTRACT

The dream of much Batch Control standardization has been the development of a common Batch language. In practice, Batch languages are superposed on collections of archaic control data bases and diagrammatic representations. A common Control Language could integrate all aspects of control. Such a language could capture the insights of continuous, logical, and batch control together, and still be measured against the various standardized Batch Models or diagrams.

The illustrated language is based on a number of ideas previously presented in individual papers: SuperVariables, Idioms, Bracket Sequencing Control Structures, and Theme Statements. As these ideas come together, they bring together the two main streams of Fluid Process Control.

## INTRODUCTION

It is normal for batch engineers to view their control as a more general and inclusive practice than continuous control. This paper will carry that view to its logical conclusion: It will present a single information model and language framework to integrate the full range of both control practices.

The framework extends earlier discussion of Idioms, SuperVariables, State Based Logic, Theme Statements, etc.[1-10] The resulting Control Language differs inherently from conventional computer languages, incorporating unique ease of use concepts, including:

- Expression of real-time sequential, parallel, and continuous computation, not just compute time sequencing.
- Structured, about the process and control organization.
- Leveled to reflect levels of human activity.
- Generating an integrated data base, for plant operation, monitoring, control, and historical recording of operating parameters and production states.
- Formatted, in semigraphic form, for readability.
- Complex hundred parameter Block control data structures replaced by one line statements; the Idioms and SuperVariables.

The earlier references address component language concepts. A short paper cannot provide an exact specification of a general Control Language. But a combination of discussion and examples, drawn from both large and small, Batch and Continuous Controls, will summarize the earlier discussion into the desired integrated framework. The discussion will alternate between batch and continuous issues, and between batch oriented control applied to continuous examples and vice versa.

This goal notwithstanding, batch and continuous perspectives differ fundamentally, requiring a rethought synthesis, an integrated perspective allowing each practice to benefit from the insights of the other. On an abstract plane the two perspectives can be summarized below:

- Continuous control is based on the continuous evaluation of all relevant aspects of the current process state, to recalculate the control actions which adjust that state for best process performance.
- Batch control is based on an assumed desired production trajectory in process variable space, on the monitoring of

events which identify the progression to later states or stages of that trajectory, and on the exercise of controls and control actions to maintain or move the process along that trajectory.

In some respects, the Batch perspective is simpler, defining the trajectory explicitly, not depending on its implied derivation from interacting state calculations. The application of a fully continuous computation to the classic Batch process would be impossibly complex. On the other hand, the accommodation of elaborate constraint or recovery controls in a batch program can create a maze of side paths in the logic. The same constraint controls can develop automatically as a consequence of an appropriate continuous design.

Often application objectives can be carried from either perspective, with interesting comparative benefits. One reference[4] contains an example in which the use of conventional continuous feedforward techniques, permits a continuous control system to take the place of three batch phases.

### The Role of Graphics

The Language design takes a radical position relative to the usual graphic panaceas. Graphics should enhance ease of use in more fundamental ways than familiarity. In fact, most control graphic diagrams were designed with a different goal than the specification of an integrated control application:

- Successful graphic diagrams abstract and simplify specialized areas of computation, and simple designs, not general computations. On the negative side, the result in process control is applications configured as a collection of weakly compatible specialized diagrams (designed from incompatible control points of view?)
- Most standard diagrams (excepting the Sequential Function Chart) were invented to reflect the connections and specifications to implement controls in a particular hardware, rather than to state the intended control objectives in the best hardware independent way.
- Often that hardware is so far out of date that many control engineers have only seen its computer based simulation. We need control representations that reflect the modern computerized control system.
- As a result, it is fair to ask if these diagrams are really the best ease of use representation for the application or the modern computer environment.

In contrast, both the text and graphics elements in the Control Language have been specifically designed to most clearly represent the intended application behavior. Specifically the graphics have the following roles (some analogous to the role of pictures in a text book).

- Text Key Words are replaced by icons (e.g. Activity Brackets), designed to reflect the underlying application structure visually.
- Icon diagrams are generated automatically (e.g. Idiom diagrams and Sequential Function Charts) to illustrate programmed text.
- Just by their inclusion, diagrams and icons can provide visual structure and landmarks to a program, cueing later searches (like looking for the text associated with a particular diagram in a book).[1]
- They are designed to reinforce the learning of a user.[2]

### Potential for Objects in Control

Objects represent a favored view of programming wherein (ideally) all application data is cataloged into data structures (the Objects), each type (Class) being defined with its access and processing functions (the Methods). The concept allows programs (including the programs which define the Methods) to connect to the data in any Object through Messages (the calls to the Method functions) to it.

The Object Class concept allows the definitions of similar application structures (the Classes) to be based on each other through a concept of Inheritance and Overriding. The strength of this capability is that it allows similar Objects to be treated similarly through Messages using the same Method names, even as the

[1] As long as the diagrams are not overworked, obscuring the program. Automatic generation of diagrams limits and standardizes their use.

[2] This reinforced learning is also the strongest benefit of the modern menu based Graphical User Interface, but only one dimension of ease of use. There many cases where it is easy to demonstrate faster, more easy use from traditional keyboards, particularly for experienced users.

system makes the low level distinctions, selecting the right Method variant for each Object. Each such Class may then be the basis of any number of working Instances, each accessed through the associated Methods.

Though newly popular for application graphic support system programming, Objects are specialized tools requiring remodeling to fit the user oriented perspective for any particular application class. Crucially, the role of Objects in an application control language model differs from the role of Objects in the software implementation of that language (where practically everything will be an Object).

The discussion takes the natural view that in a Process Control model the Process elements will be Objects, all procedural aspects then being treated as Methods, invoked through Messages. On the other hand, general control functions and concepts used to build the control system will also be Objects, made live by their own Methods and Messages. Table 1 summarizes the result.

**Table 1:**

| Category | Objects | Methods | Messages/ Calls |
|---|---|---|---|
| Process | Process Units and Elements | Control Tasks, Batch Tasks/ Phases, Recipe Procedures | Task/Phase Calls, Recipe Calls[a] |
| Control | Control Block Function/Concept (e.g. PID) | Control Algorithms | Control Block Declarations[b], Data Accesses |

a. The preprogrammed Recipe is like a programmed Message even though it must be supported by a permanent record. (This record may be implemented in software as an Object or a File. But from an application point of view it is still a Message.

b. As calls on the control Object, these are Messages, but Messages which list the entire Block Object contents. The Block Parameters constitute the Message argument list. In this sense, the Block Object and the Block Message are almost the same!

In the Language, all of these Objects are defined within the single procedural framework. But there are additional Object related Language requirements:

- The potential complexity of Formulae with large numbers of Parameters requires a specialized support structure, like the corresponding Block support structure. This must allow Recipes and their Formulae to be created operationally, and separately named and manipulated.[3]
- Modeling of one Process element on another. This is the normal role of Object Class definition and instantiation. The Language makes no distinction between Class and Instance Objects; the user can model any Process Operation (Object)[4] on any already defined Operation (whether concrete or not). Class and Instance distinctions can be made as a matter of convenience.[5]
- Modeling of Process Objects in terms of lower level component (Process, Equipment, or Control) Objects; this provides an Object formalism with two hierarchies: A hierarchy of component Objects and a Hierarchy of inheritance and modeling.
- The Plant equipment is relatively permanent; accordingly the basic Process element and component Objects are configured as permanent elements of the control system. But the associated control Objects may be active or inactive, turned on or off, or Called to be active.
- Modeling of temporary Process Lines or combinations, and accommodation of their processing records. A concept of temporary Objects permits these Lines to be treated like any of the permanent process combinations. This requires that the temporary component Objects be selected by arguments defined only in the activating Message. Like any other argument, these will be restricted according to type. There may be several similar such Process Lines, each with its assignable argument Objects. But since the equipment bottlenecks will be well defined, the corresponding temporary Objects can be predefined, even as their components will change with use.
- The temporary Object allows the kind of sharing of equipment where a process element is committed for the duration of a product batch. But there are two other kinds of shared equipment needed for batch:
  - Booking (of an Operation or Variable) for which all internal data can be read by anyone, but changed only by the Booking Operation. This is a more flexible kind of sharing, but still with restrictions.

[3] As implemented, these Messages would be represented as special kinds of information Objects, stored internally for deferred use. Similar Objects would keep record of the batch. These would also be Messages from an application point of view: Messages to some display.

[4] The term Operation, after Unit Operation, is used in the language to represent the Language counterpart of a Process element or Object.

[5] Distinct Class Objects become more important if the associated language supports in program creation of the objects.

- Freezing (of an Operation or Variable) which permits setting and Freezing by any later sharing Operation as long as the setting agrees with the setting made by the initial Freezing Operation. This is the kind of sharing needed to support blocking valves, where separate valve action requests represent the same interest in blocking any contamination.

# BASIC CONTROL ELEMENTS

Most of the language elements will be described in application examples below. But a few of the language techniques are distinct enough to need their own separate introduction. Particularly distinct are the Control Idiom and State Based Logic concepts.

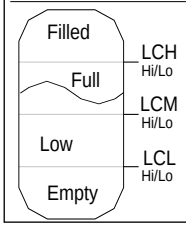
## Logic Based on States

Logic is basic to batch control, having led to the early batch application of digital logic controllers. Early logic controller users debated the use of ladder versus logic diagrams. In earlier presentations[5], the author has demonstrated the superior readability of State based ladder diagrams and truth tables over conventional boolean logic based IF statements. The Control Language carries out all logical control in terms of named States, whether system or user defined, each associated either with a variable or higher language functional element.[6]

Going further, logic diagrams allow the tracing of logical signals, whereas ladder diagrams and truth tables allow the tracing of cause and effect between states and events. As an event oriented activity, batch is best addressed from the latter point of view; for the actual batch application, the concept and choice of logical signal is an artifact of the control implementation.[7]

Control logic should be expressed in a hierarchy of levels of meaningful process/processing states, rather than in terms of Boolean truth values. Figure 1 shows how straightforwardly the truth table manages to relate the computed States of a tank Level variable to underlying level contact variable States, comparing it to the corresponding IF statement clumsiness.

**Fig. 1. Truth Tables vs. If Statements**




| LCH | LCM | LCL | Level |
|---|---|---|---|
| | Lo | Lo | Empty |
| Hi | Lo | Hi | Low, Failed |
| | **Lo** | **Hi** | **Low, OK** |
| Lo | Hi | Lo | Full, Failed |
| Lo | Hi | | Full, OK |
| Hi | Hi | | Filled |

```
IF(LCL=Lo) {IF(LCM=Lo) THEN Empty
            ELSE IF(LCH=Lo) THEN Full,Failed}
ELSE IF(LCM=Lo) {IF(LCH=Hi) THEN Low,...
                 ELSE Low}
ELSE {IF(LCH=Lo) THEN Full
      ELSE Filled}
```

Instead of IF statements, the Control Language uses State based Case Statements,[8] in which any necessary logical computations are carried out in a separate associated truth table, as shown in Figure 2.

**Fig. 2a. Related Conditional Statements**

```
PROCESS. TANK                                  Page: Procedures

TEST_TANK
  [LCH; LCM; LCL];
    LOW/EMPTY: TKV, OPEN;
◊   FULL: END;◊
    FILLED: TKV, CLOSE
```

**Fig. 2b. Supporting Truth Table**

PROCESS. TANK  Page: Details

| LCH | LCM | LCL | Result |
|---|---|---|---|
| | LO | LO | EMPTY |
| HI | LO | HI | LOW, Failed |
| | LO | HI | LOW, OK |
| LO | HI | LO | FULL, Failed |
| LO | HI | | FULL |
| HI | HI | | FILLED |

In Figure 2a, the logical cases are based on States of those level contacts, declared by enclosing LCH, LCM, and LCL in square brackets. Three State Prefixed statements then reflect the different actions to be taken, depending on the computed State. States reflected in these Prefixes, but, as above, not returned directly by the

[6] The States need not be binary valued. States also have the interesting property of self-naming their own data field, not requiring some structure element name.

[7] That is, the States of the batch production could be grouped under many different collections of State valued variables without affecting the logic.

[8] As indicated later, basic control computations are expressed on a Procedures Page; a Details Page is generated for any necessary supporting computations.

contacts, are computed in an automatically associated Truth Table (Figure 2b).

This allows the clean separation of the logical computations from their programmed consequences, whose usual inter-tanglement is a common source of IF statement error. The essence of the control action is still centered and clear in the basic Procedures Page.

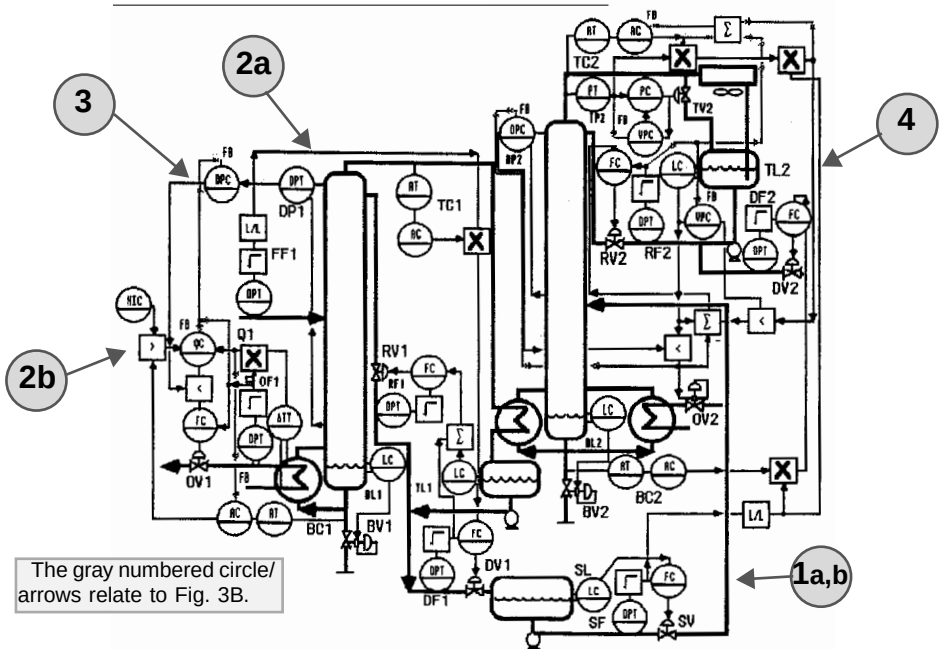
Whenever possible, the State Prefix consists of a single State name representing the State for which the action is to take place. However, the Prefix allows for the inclusion of ORed State conditions as shown in the first (LOW/EMPTY) State Prefixed Statement above. Alternatively the Prefix may included ANDed States (LOW,EMPTY) or States preceded by a special Modifier (such as NOT).

As illustrated later, the language supports other simplifying special cases including Real data based States. The above example also shows a State assignment (e.g. TKV, OPEN) for simple setting of variable or system States.

## Continuous Control and Idioms

Control Idioms are designed to clarify the intent of continuous control designs. Figure 3 below introduces them in a complex continuous process example. Their special benefits to batch application can then be presented. Traditionally the continuous control design would be defined by the block diagram (Figure ). Instead, the Control Language represents the design as a Procedures Page listing of Idiom Loop Statements (Figure 3b).

**Fig. 3a. Continuous Control Block Diagram**



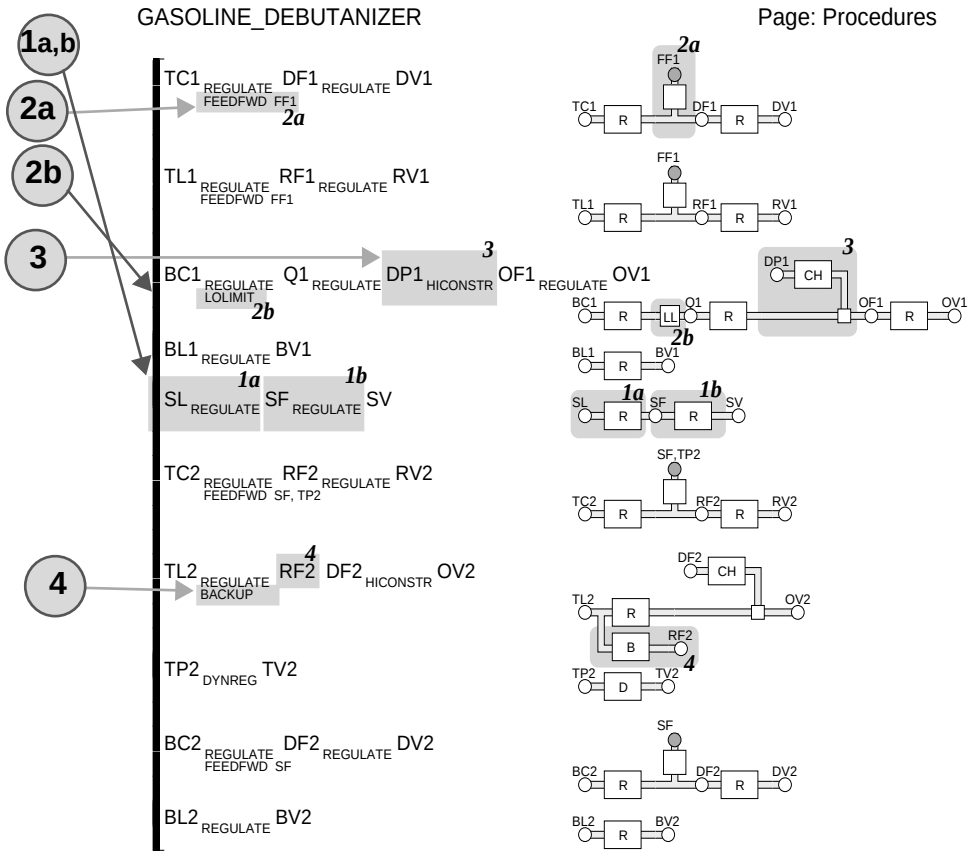

Each Statement describes a generalized process control loop, starting on the left of the page by specifying the primary variable, followed by subscripts specifying the desired control intent applied to that variable. The statement continues similarly with secondary variables, with their control intent, terminating at the right of the page with the final manipulated valve(s). For example, the top line specifies Regulation of the Top Composition of the first tower (TC1), with feedforward accommodation of the tower's Feed Flow (FF1), driving a flow loop Regulating the Distillate Flow (DF1), which finally manipulates the Distillate Valve (DV1).

The overall statement form is structured in terms of the four special component patterns below, indicated in the figure text by the numbered gray rectangles, and in the icons by the numbered rounded rectangles:

**1.** The first pattern (e.g. $SL_{REGULATE}$) represents the basic (PID or other) regulation of its included variable (SL). The structure is

**Fig. 3b. Idiomatic Loop Statement Representation**




always followed by continuing structures which may include repetitions of the same structure (e.g. $SF_{REGULATE}$), in cascade, or final valve variables (SV in the example).

2. The basic Regulate structure can be modified by secondary Idioms which support, improve, or constrain its action without changing its basic regulatory function (e.g. FEEDFORWARD (2a), LOLIMIT (2b), or HILIMIT).
3. The Loop Statement can modify the impact of preceding structures by adding Hi or Lo Constraint Override structures (e.g. $DP1_{HICONSTR}$).
4. The Loop can include multivariable controls, the simplest of these being fanout structures, illustrated here in a Backup control defining an alternative regulatory path to be used when the main regulation path has been overridden.

The Idiom and its Loop Statement simplify the specification of the more complex continuous usages, such as feedforward and constraint control. For the batch practitioner, this makes them more accessible. But a single line Idiom Loop statement also allows continuous controls to be combined into conventional computations, permitting conditional or sequenced changes in control structure. Such a capability is especially suited to batch control and sequenced start-ups and shutdowns of all processes, as illustrated below:[9]


The figure shows an alternative version of the earlier PROCESS.TANK TEST_TANK Task with a conditionally operated control structure. In support of this kind of capability, Idioms have been implemented in a way that permits smooth (bumpless) transition between consecutively applied Loop Statements or complete control structures, independent of detailed character.

[9] Each Idiom has an associated Parameters Page parameter record like a conventional Block.

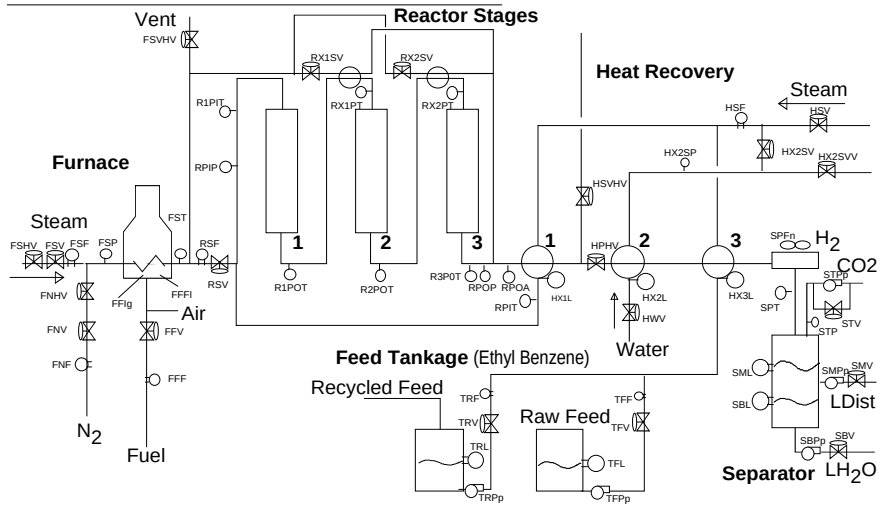
 

# COMPLEX AUTO-STARTUP IN A MULTI-UNIT PLANT

This section uses complex startup/shutdown in a continuous multi-unit plant to illustrate the basic Language sequencing. The following section will raise special batch sequencing concerns. A continuous Styrene Process[12-13] mixes ethyl benzene and superheated steam at temperatures approaching 700˚C and passes them over a catalyst bed to produce Styrene (plus $H_2$ and $CO_2$). The steam serves three purposes: it heats the reaction, it buffers the styrene molecules to prevent their polymerization, and it prevents liquid water from poisoning the catalyst.

The plant (Figure 4) includes five units: Furnace, Reactor, Heat Recovery, Separator, and Feed Tankage.

**Fig. 4. Styrene Multi-Unit Process**




A simpler, more reversible process might be started automatically by starting all parts independently; this process requires coordinated (batch style) event sequencing of all stages of operation.

In more detail, Ethyl Benzene is pumped from the Feed Tankage, initially vaporized in the Heat Recovery exchangers, combined with the steam, further heated in exchangers, combined with superheated steam from the Furnace, reacted in the Reactors, cooled in the Heat Recovery exchangers, condensed in the air condenser, and separated into $H_2$, $CO_2$, $H_20$ and distillable product.

The Reactors must be purged of oxygen (explosion hazard with $H_2$). The Furnace must be heated slowly, not to crack its walls. The initial water free purge and heating is carried out under a $N_2$ atmosphere.

## Process Nomenclature

The table below lists the process variables, generally coding each in terms of the associated process unit, the particular stream, an inlet/outlet differentiator, and the variable type.

**Global Styrene Plant Variables**

**FST:** Furnace Steam Temperature
**TFF:** Tankage Feed Flow
**POL:** Product Outlet Tank Level
**TRF** Tank Recycle Flow
**TF=TRF+TFF:** Tankage Flow
**WOL:** Water Outlet Tank Level

**Furnace Variables (See Global also)**

**FSHV:** Furnace **Steam** Hand Valve
**FSF:** Furnace Steam Flow
**FSV:** Furnace Steam Valve
**FSP:** Furnace Steam Pressure
**FNHV:** Furnace **Nitrogen** Hand Valve
**FNF:** Furnace Nitrogen Flow
**FFF:** Furnace Fuel Flow
**FFFI:** Furnace Fuel Flame
**FNV:** Furnace Nitrogen Valve
**FFV:** Furnace **Fuel** Valve
**FFIg:** Furnace Fuel Ignition
**FSVHV:** Furnace **Steam Vent** Hand Valve

**Reactor Variables**

**RSV:** Reactor **Steam** Valve
**RSR=RSF/TF:** Reactor Steam Ratio
**RSF:** Reactor Steam Flow
**RX1SV:** Reactor **Exchanger** 1 Steam Valve
**RX1PT:** Reactor Exchanger 1 Product Temperature
**RX2SV:** Reactor Exchanger2 Steam Valve
**RX2PT:** Reactor Exchanger 2 Product Temperature
**RPIT:** Reactor **Product** Inlet Temperature
**R1PIT:** Reactor 1 Product Inlet Temperature
**R2POT:** Reactor 2 Product Outlet Temperature
**RPOP:** Reactor Product Outlet Temperature
**RPIP:** Reactor Product Inlet Pressure
**R1POT:** Reactor 1 Product Outlet Temperature
**R3POT:** Reactor 3 Product Outlet Temperature
**RPOA**: Reactor Product Outlet Analysis

**Heat Recovery Variables**

**HSV:** Heat Recovery **Steam** Valve
**HSF:** Heat Recovery Flow
**HX1L:** Heat Recovery Exchanger 1 Level
**HX2L:** Heat Recovery Exchanger 2 Level
**HX2SV:** Heat Recovery Exchanger 2 Steam Valve
**HX2SP:** Heat Recovery Exchanger 2 Steam Pressure
HX3L: Heat Recovery Exchanger 3 Level
HSR=HSF/TF: Heat Recovery Steam Ratio
**HWV:** Heat Recovery **Water** Valve
**HSVHV:** Heat Recovery Steam **Vent** Hand Valve
HSX2VV: Heat Recovery Exchanger 2 Steam Vent Valve
HPHF: Heat Recovery Product Hand Valve

**Feed Tankage Variables (See Global also)**

**TFL:** Tankage **Feed** Level
**TFV:** Tankage Feed Valve
**TRL:** Tankage **Recycle** Level
**TRV:** Tankage Recycle Valve
**TFPp:** Tankage Feed Pump
**TRPp:** Tankage Recycle Pump

**Separator Variables**

**SPFn:** Separator **Product** Cooling Fan
**SPT:** Separator Product Temperature
**STV:** Separator **Top** Valve
**STP:** Separator Top Pressure
**STPp:** Separator Top Pump
**SMV:** Separator **Mid** Valve
**SML:** Separator Mid Level
**SMPp:** Separator Mid Pump
**SBV:** Separator **Bottom** Valve
**SBL:** Separator Bottom Level
**SBPp:** Separator Bottom Pump

## Control System Structure

The Language models a process hierarchically in terms of its Operations/Objects, modeling the process divisions: the Styrene Plant Operation and the Furnace, Reactor, Heat Recovery, Separator, and Feed Tankage SubOperations. Each Operation is organized into Pages for modeling distinct control functionalities.[10] The example shows several of these Pages[14], mostly Procedures Pages, describing active control procedures.

### Control Tasking and Sequencing

A statement language must list its statements in distinct execution orderings. These include: Sequenced and Looping order (as in conventional computer languages), or (as further required by real-time control activities) Parallel, Continuous (individually repeated each sample time), and State Driven. The language uses iconic brackets (which may be nested) to distinguish execution orderings.

Figure 5 illustrates the usage. The diamond icons define exits. The icons replace keywords, more clearly highlighting program structure. As illustrated later, the icons allow all the usages of a Sequential Function Chart, and then some.

When the Styrene Plant Operation is called, the plant is activated in a simple sequence. The outer (left most, thin lined) bracket represents this sequenced Activity, which first tests that all of the lower level units (SubOperations) have been shut down, prior to executing the following nested Activities. The outermost of these Activities (bracketed with two parallel lines) encloses two other Activities, to be carried out in parallel.

In turn, the first executes continuously (represented by a solid continuous bracket), monitoring an operator console shutdown button state (Shutdown or Emergency Shutdown). The second (a dashed bracket) represents a state driven Activity, selecting a lower level Activity, depending on the current (Startup, Hold,

[10] The earlier discussions show Procedures Pages; the above described global (real valued) Styrene Plant variables are defined on a Definitions Page:

STYRENE_PLANT Page: Definitions

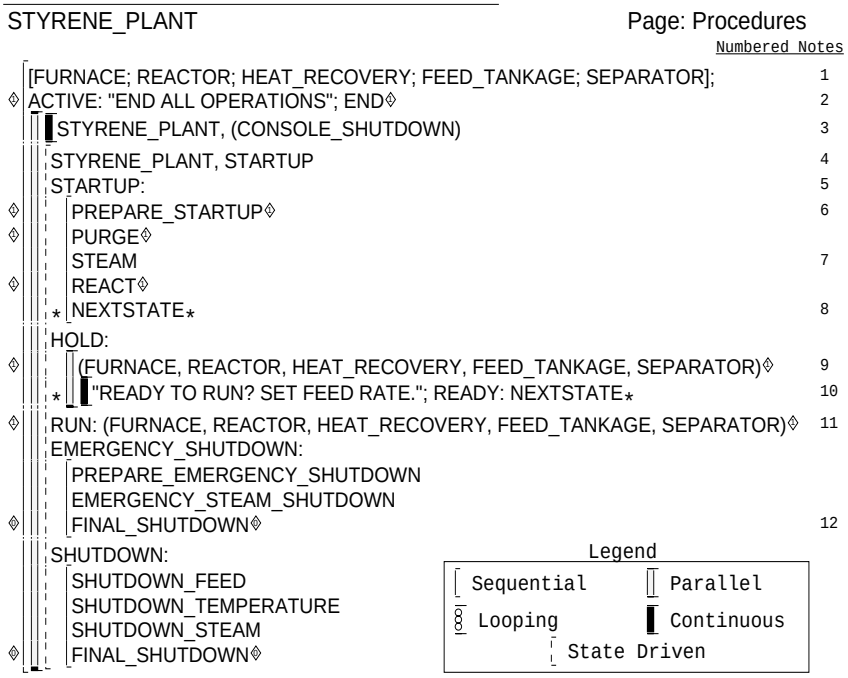
| NAME | IN | VALUE | MIN | MAX | UNITS | SET | HI | LO | DEV |
|---|---|---|---|---|---|---|---|---|---|
| FST | 1 | _ | 0 | 800 | DGC | _ | 780 | 300 | 20 |
| TRF | 2 | _ | 0 | 100 | GPM | _ | _ | _ | 20 |
| TFF | 3 | _ | 0 | 100 | GPM | _ | _ | _ | 20 |
| TF | _*1 | _ | 0 | 100 | GPM | _ | _ | _ | 20 |
| POL | 4 | _ | 0 | 300 | FT | _ | 30 | 10 | 20 |
| WOL | 5 | _ | 0 | 300 | FT | _ | 30 | 10 | 20 |

It groups process variables under a common, user defined, SuperVariable heading structure[7], replacing rigid I/O blocks of current practice with readable, free form, language structures. For example, the **VALUE** heading makes its associated Attributes real valued, distinguishing the variables as Real; replacing it by a **STATE** Attribute heading would make these Discrete variables. Note the **TF** related Footnote reference to a Details Page defining computation:

TYRENE_PLANT Page: Details

*1 TF = TRF+TFF

**Fig. 5. Activity Brackets in the Main Procedures Page**




Run, Shutdown, or Emergency Shutdown) system State. Normally, this activity proceeds through the Startup, Hold, and Run States to startup and run the plant.

The Startup State corresponds to a sequencing through the necessary Tasks: Prepare Startup, Purge (clearing oxygen from the system; slowly heating the furnace under Nitrogen), Steam (transitioning from a Nitrogen process atmosphere to a Steam atmosphere), and React (initiating a low Feed and reaction).

Once the reactor is in operation, the State is changed to Hold, all Units called (activated), and the intended final feed established. The system enters the Run state; operation continues normally until operator intervention calls for Shutdown or Emergency Shutdown.

The Emergency Shutdown consists of the sequenced Tasks: Prepare Emergency Shutdown (removing the feed from control and waiting 10 minutes for settling), Emergency Steam Shutdown (rapid steam rampdown, replacing with Nitrogen), and Final Shutdown (abruptly closing process valves).

The normal Shutdown consists of a more orderly sequencing of shutdown tasks: Shutdown Feed (ramping the feed down), Shutdown Temperature (ramping the temperature down to 350° over two hours), Shutdown Steam (ramping the steam down, replacing it with Nitrogen), and Final Shutdown.

### Detailed Plant Level Notes

This section explains the language details according to the note numbers in the main Activity:

1. The States of all subOperations are to be considered in the following State Prefix test.
2. This statement is executed if any unit (subOperation) returns the **ACTIVE** State, indicating a Unit still operating. In that case, the message is sent to the operator and the entire Activity terminated.
3. This Continuously operating State assignment sets the Styrene Plant Operation from the Console Shutdown variable (affecting operation when there is a change in State).
4. The State Driven Activity starts executing the first statement which then sets the Startup State.
5. The Startup State Prefix applies to the following (Startup) Sequential Activity.
6. Normal termination from the PREPARE_STARTUP Task Call continues operation with the next statement. Abnormal exit (the diamond) terminates control of the entire Activity (to the level specified by the diamond).
7. This Task call terminates normally only, continuing execution in the next statement.
8. The **NEXTSTATE** command advances the State to the State of the next Prefix.
9. Assignments and calls can include lists; in this case: **(FURNACE, REACTOR, HEAT_RECOVERY, SEPARATOR, FEED_TANKAGE)**, which Calls all Activities. The terminating diamond indicates termination of the entire Activity by any

failed Operation.

**10.** A continuous, single statement Activity terminated by operator action; a more direct statement of the intent of a statement which would otherwise be expressed as a program loop.

**11.** The normal State conditioned statement with its prefix, followed by a statement which continues all control.

**12.** The normal Shutdown termination.

## Plant Level (Global) Tasks

This section lists the first two Tasks (still part of the main plant Operation) called in the above listing:

### Prepare Startup Task; Outline Translation

```
STYRENE_PLANT                                                    Page: Procedures
                                                                   Numbered Notes
PREPARE_STARTUP                                                                 1
  IN FURNACE: (FSHV, FNHV, FSVHV, FSV, FNV, FFV), CLOSE                         2
  IN REACTOR: (RSV, RX1SV, RX2SV), CLOSE
  IN HEAT_RECOVERY: (HPHV, HSVHV, HSV, HX2SV, HX2SVV, HWV), CLOSE
  IN FEED_TANKAGE:                                                              3
    (TRV, TFV), CLOSE
    [TRL, 20; TFL, 20];                                                         4
◊   LO: "INADEQUATE FEED"; END◊                                                 5
  IN SEPARATOR:
    (STV, SMV, SBV), CLOSE
    [POL, 10; WOL, 10];
◊   HI: "INADEQUATE PRODUCT STORAGE"; END◊
```

This Task closes valves in the Furnace, Reactor, and Heat Recovery Operations, to configure the process for Startup. It tests Feed Tankage levels for adequate feed and Product tankage levels for adequate capacity.

### Prepare Startup Task; Numbered Notes

**1.** A Task is a Named (above the Bracket) Activity.

**2.** Statements within an Operation's Activities or Tasks can normally affect only the directly referenced elements within the current Operation. But a Scoping Prefix allows general access to any directly referenced SubOperation's Tasks or variables. Together with the List State assignment, it allows the closing of the listed Furnace valves, compactly and clearly stated. The alternative to the Scoping Prefix usage would, in this case, be a statement like:

**(FURNACE.FSHV, FURNACE.FNHV, FURNACE.FSVHV, ... , FURNACE.FFV), CLOSE**

**3.** This Scoping Prefix is applied to an entire Activity.

**4.** This Declaration includes real variables, each with a comparison limit; the comparisons each return one of the States: **LO/ EQUAL/HI**.

**5.** If either comparison in the above Declaration returns a State matching the **LO** State Prefix (insufficient Feed), the Task terminates.

### Purge Task; Outline Translation

This Task sequences, controlling the Heat Recovery Exchanger 2 (Level and Pressure), supporting initial product cooling and feed steam generation. Furnace, Reactor, and Heat Recovery Steam Vent are connected by opening valves. The remaining steps are carried out in the Furnace. Nitrogen is placed under control and ramped to 5 CFS (in 2 minutes). Fuel ignition is started, and fuel ramped to 5% valve position.

```
STYRENE_PLANT                                                    Page: Procedures
                                                                   Numbered Notes
PURGE
  IN HEAT_RECOVERY: CONTROL_HX2                                                 1
  IN REACTOR: RSV, OPEN
  IN HEAT_RECOVERY: HSVHV, OPEN
  IN FURNACE:
    CONTROL_NITROGEN                                                            2
    RAMP FNF FROM 0 TO 5 CFS IN 2 MIN                                           3
    FFIg, START
    RAMP FFV FROM 0 TO 5% IN 2 MIN
    FFIg, STOP
    [FFFI] COLD:
      "IGNITION FAILURE"
      FFV, CLOSE
◊     END◊                                                                      4
    CONTROL FURNACE TEMPERATURE
    "FURNACE TEMPERATURE RAMPING TO 650, MONITOR R3POT."
    RAMP FST TO 650 IN 6 HR                                                     5
```

The igniter is stopped and the flame tested. If cold, the fuel valve is closed and the Task terminated. Otherwise, the Furnace Temperature is placed under control and ramped to 650˚ (in 6 hours), the operator so notified.

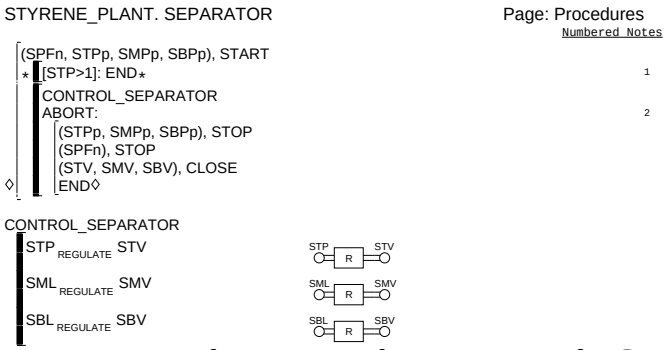
 

### Purge Task; Numbered Notes

1. This statement initiates a continuous control Task[11] in the Heat Recovery Operation, allowing that Activity to continue execution without hanging up, until the Purge Task terminates.
2. A Continuous non-terminating control Task initiated (as in 1), running in the Furnace Operation.[12] These cases illustrate the sequenced evolution of the control structure throughout the program by successive calls of control Tasks.
3. Ramps[9] the already controlled Furnace Nitrogen Flow to 5 cubic feet per second in 2 minutes.
4. END statement and diamond: an abnormal termination.
5. Without From phrase, ramping starts at current value.

### Separator Operation Procedures

Like the Furnace SubOperation Procedures listed in Footnote 12, the Separator Procedures Page defines the basic controls of its SubOperation. The Separator Activity is more interesting; it starts pumps and tests/




waits for the Separator Top Pressure to reach one atmosphere to initiate the Control Separator Task. It then redefines a tailored **ABORT** response, supporting an orderly shutdown whenever the Task is aborted from the outside.

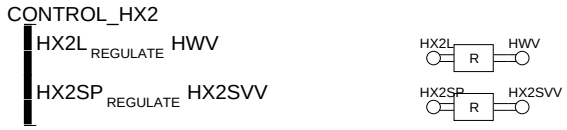
### Separator; Numbered Notes

1. Shorthand comparison; **END** terminates the immediate Activity only.
2. **ABORT** is a system State entered whenever the Task or Operation is terminated abnormally. A State Prefix keyed on it expresses a statement or Activity to be carried out before the completed Task or Operation termination, in this case the orderly shutdown of the Separator.

## BATCH CONTROL REQUIREMENTS ILLUSTRATED

Figure 6 shows a multi-unit Batch Reactor system capable of running two independent batches simultaneously, while sharing equipment between the productions:

[11] A two loop Heat Recovery subOperation Task:




[12] These are the Furnace subOperation Control Tasks, several defined in single line statements:

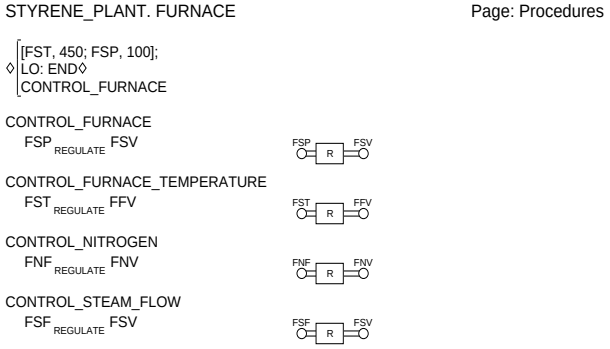
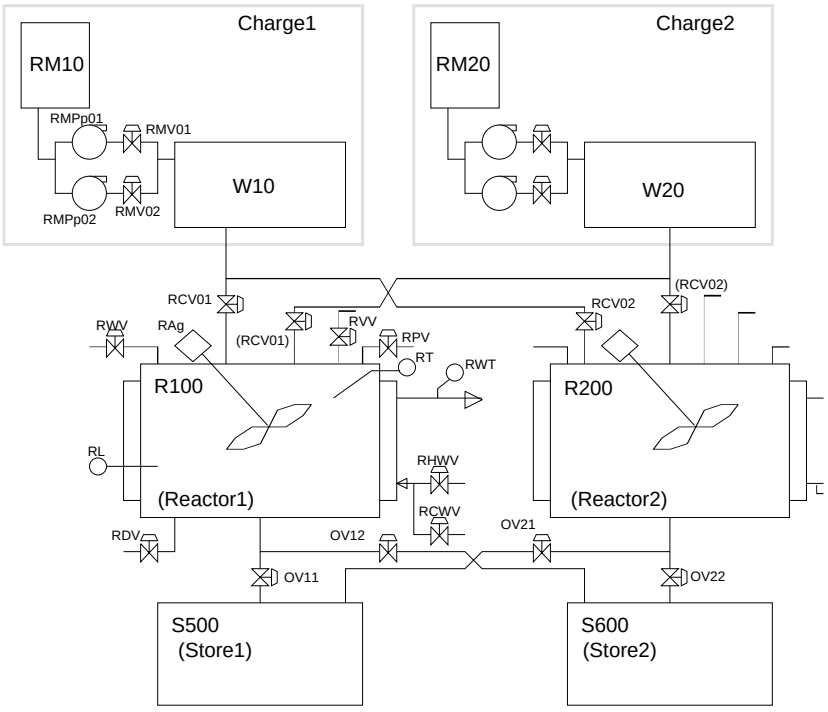


**Fig. 6. Multi-Unit Batch Process**




Both Charging Units (and their associated Charge subOperations) may be used to charge either Reactor. Either Store can be used to store the product from either Reactor. One complication comes in the programming of the irregularly arranged charging and blocking valves.

The Operation Page summarizes the structure of an Operation: System and User defined States, Tasks, and SubOperations.

**Fig. 7a. Batch Plant Operations Page**

BATCH_PLANT Page: Operation

SYSTEM STATES: CONFIGURE/SETUP/SIMULATE/OPERATE,
RUN/SUSPEND/CONTINUE/ABORT/END,
ACTIVE/ INACTIVE, BOOKED/UNBOOKED, _/INITIALIZE

USER STATES: _/RECIPE_RUN/RECIPE_HOLD/RECIPE_DONE

TASKS:

SUBOPERATIONS: Charge1, Charge2(Charge1), Reactor1, Reactor2(Reactor1),
Store1, Store2(Store1), BATCH_TRAIN:[2]

**Fig. 7b. Batch Train Operations Page**

BATCH_PLANT. BATCH_TRAIN: [2*] Page: Operation
(ReactorA*(Reactor1): (Reactor1/Reactor2), Load1, Load2, Water_Load, StoreA(Store1): (Store1/Store2))

SYSTEM STATES: CONFIGURE/SETUP/SIMULATE/OPERATE,
RUN/SUSPEND/CONTINUE/ABORT/END,
ACTIVE/ INACTIVE, BOOKED/UNBOOKED, _/INITIALIZE

USER STATES: RUNNING/HOLD_PHASE/NEXT_PHASE

TASKS: CHECK_BOOKING, TRANSFER_RECEIVE

SUBOPERATIONS: ReactorA, StoreA

Figure 7a shows eight subOperations:

- Charge1 and Charge2 (modeled after Charge1); the modeling is indicated by the parenthesized expression: Charge2(Charge1).
- Reactor1 and Reactor2(Reactor1) (i.e. modeled after Reactor1).
- Store1 and Store2(Store1) (i.e. modeled after Store1).
- BATCH_TRAIN:[2], indicating a basic temporary Operation with at most two instances active at a given time. This is like an Array of BATCH_TRAIN Operation/Objects.

For each of the Model Operations (Charge1, Reactor1, Store1), the user would configure the usual defining Pages normally. The affect of each of the Modeling Operation declarations (Charge2(Charge1), Reactor2(Reactor1), Store2(Store1)) is to default the Modeled Operation's Page declarations, to be identical to its Model Operation's Pages. Any variations (notably variable I/O hardware addresses) must then be configured to override the defaults.

Figure 7b corresponds to the temporary BATCH_TRAIN Operation. The top line includes the Operation name, with square bracketed 2, echoing the usage above. In this case the 2 is followed by an asterisk, keying the

ReactorA name in the second (argument list) line.

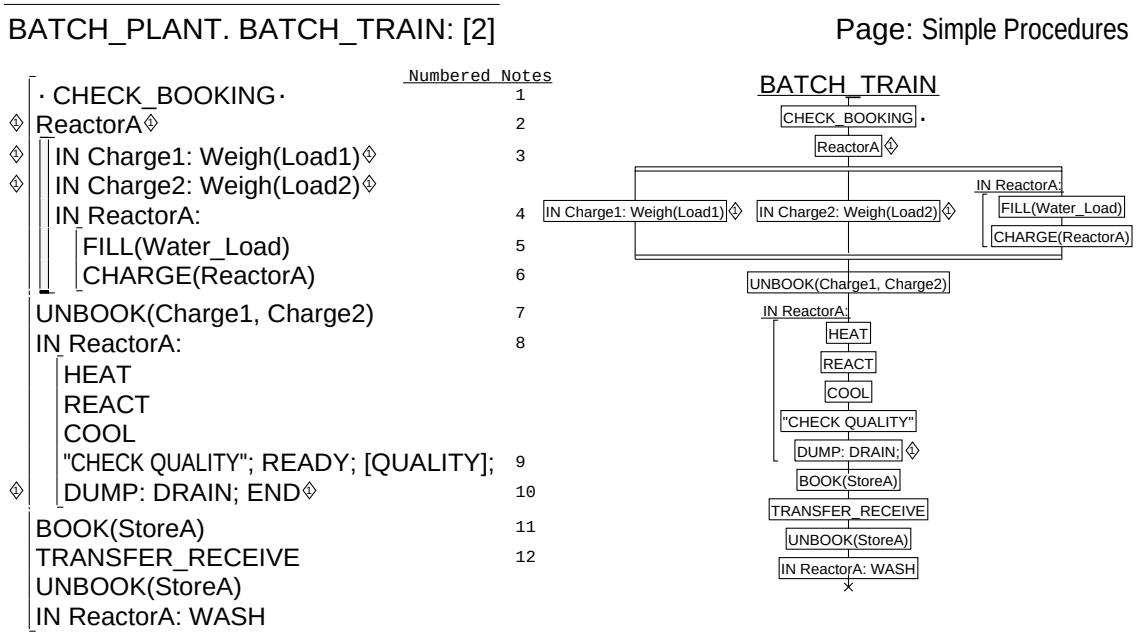
This signifies that the ReactorA argument is the bottle-neck operation defining the limitation to the number of simultaneous running Trains; once each Reactor has been assigned as the ReactorA in a running Train, no more (temporary) Trains are possible. The argument list also contains arguments for: the Load quantities to be loaded from each charge unit, how much water to load, and the choice StoreA of Store unit.

More specifically, the expression ReactorA(Reactor1): (Reactor1/Reactor2) means that the argument reactor ReactorA must be modeled on Reactor1, and chosen from Reactor1 or Reactor2. The expression StoreA(Store1): (Store1/Store2) is similar.

The argument list constitutes the Recipe Formula for the Batch Train. Because real Formulae can have many, complex arguments, the language must support the management of such lists. It treats them similarly to conventional control Block parameter records, but allows a simpler operator appropriate MAKE_RECIPE command to create the Formulae for an already written Batch Procedure.

As in the earlier example, any Operation may have an associated un-named Activity, which acts whenever the Operation is called. In this case, this Activity (Figure 8a) acts as the basic Batch Train Recipe Procedure/Task.

**Fig. 8a. Batch Train Recipe Procedure/Task**




In this case, a Simple Procedures Page Recipe Language variant is used, based on a restricted subset of the Control Language's procedural statements. This subset is designed for simpler programming of Recipe Procedure structures from calls to already defined detailed Tasks. It can support the automatic translation to an illustrating Sequential Function Chart, as shown.

The Recipe calls ten such detailed Tasks, most being self evident. The more complex Tasks manage the Booking of Charge Operations and the Freezing of the blocking valves to the Stores, as shown in Figure 8b.

If Process hardware lent itself to more mathematically pure representation we could handle the blocking valves in some array-like structure. In fact, the Language needs methods to cleanly indicate the different cases. The Language does this above by allowing the names assigned as arguments to act as State names of their corresponding dummy arguments, and to thus distinguish the different possible cases.

**Fig. 8b. Batch Train Recipe Support Tasks**

```
BATCH_PLANT. BATCH_TRAIN: [2]                    Page: Procedures

CHECK_BOOKING
  [ReactorA; StoreA] Reactor1, Store1:
    [BOOK(ReactorA, Charge1, Charge2, OV11); FREEZE(OV12, CLOSE; OV21, CLOSE)]
.   WAIT NONE FAIL; EXECUTE·
  [ReactorA; StoreA] Reactor1, Store2:
    [BOOK(ReactorA, Charge1, Charge2, OV12); FREEZE(OV11, CLOSE; OV22, CLOSE)]
.   WAIT NONE FAIL; EXECUTE·
  [ReactorA; StoreA] Reactor2, Store1:
    [BOOK(ReactorA, Charge1, Charge2, OV21); FREEZE(OV11, CLOSE; OV22, CLOSE)]
.   WAIT NONE FAIL; EXECUTE·
  [ReactorA; StoreA] Reactor2, Store2:
    [BOOK(ReactorA, Charge1, Charge2, OV22); FREEZE(OV12, CLOSE; OV21, CLOSE)]
.   WAIT NONE FAIL; EXECUTE·
  (Charge1, Charge2), START

TRANSFER_RECEIVE
  [ReactorA; StoreA];
    Reactor1, Store1:  OV11, OPEN; WAIT 5 MIN; OV11, CLOSE;
    Reactor1, Store2:  OV12, OPEN; WAIT 5 MIN; OV12, CLOSE;
    Reactor2, Store1:  OV21, OPEN; WAIT 5 MIN; OV21, CLOSE;
    Reactor2, Store2:  OV22, OPEN; WAIT 5 MIN; OV22, CLOSE;
```

# CONCLUSIONS

For many years the batch industry has sought a standard batch language. In fact, the broader process control need is for fewer languages and data bases per application, as much as possible integrating every application into a single language framework. Thus all of Process Control would be better served if the field could arrive at a single Control Language.

The paper has illustrated the possibilities of such a single Language. The proposal is based on specific concepts of ease of use and the relationships of the different elements of control computation. It casts many of the standard Batch Model concepts into an extended Object Oriented framework.

By unifying, simplifying, and clarifying the handling of all aspects of control, such a language could put the definition, documentation, or reading of a control program within the hands of any single user. This paper has shown how the control of several excerpted continuous and batch examples are unified by such a framework.

# Bibliography


[1] E.H. Bristol, “Small Languages for Large Systems and Other Uses”, ‘79 JACC, Denver, Jun. 1979.

[2] E.H. Bristol, “Strategic Design: A Practical Chapter in a Textbook on Control”, ‘80 JACC, San Francisco, Aug. ‘80.

[3] A. Prassinious, T. McAvoy, E.H. Bristol, “A Method for the Analysis of Complex Control Schemes”, ‘82 ACC, Arlington VA, Jun. ‘82, pp. 1127–1132.

[4] E.H. Bristol, B.D. Campbell, A. Gunkler, “A Batch Language Study”, ISA 81 Fall Conference, Anaheim CA, Oct. 6–8, ‘81, Paper C.I. 81–797, pp. 394–405.

[5] E.H. Bristol, “Standardizing Application Language Systems”, Chemical Engineering Progress, Aug. ‘84, 65–70.

[6] E.H. Bristol, “Computer Language Structure for Process Control Applications, and Translator Therefore”, U.S. Patent No. 4,736,320, Apr. 5, ‘88.

[7] E.H. Bristol, “SuperVariable Process Data Definition”, ISA SP50.4 Working Paper, Oct. 24, ‘90.

[8] E.H.Bristol, “Rules, Statements, and Idioms”, 17th Annual Advanced Control Conference, Purdue University, West Lafayette, IN, Sept. 30 – Oct. 2, ‘91.

[9] E.H. Bristol, “Sequencing, Logic, and Theme Statements”, Annual AIChE Meeting, Chicago, Nov, ‘91.

[10] E.H. Bristol, “Even the PID Needs Theory/Software Advances”, Control Engineering Expo and Conference, Mar. 8–9, ‘93.

[11] A. Ghosh, “Why Batch Process Control is not Chemical Engineering”, Annual AIChE Meeting, Chicago, Nov, ‘91.

[12] “Styrene”, ‘93 Petrochemical Handbook, Hydrocarbon Processing, Mar. ‘93, p. 212.

[13] J. Bernard and A. Gunkler, Computer Strategies for the Fluid Process Industries, ISA, 1990, pp. 172–174.

[14] E.H. Bristol, “A Language for Integrated Process Control Application”, Retirement Symposium in Honor of Prof. Ted. Williams, Purdue University, West Lafayette, IN, Dec. 5 – 6, ‘94.