A Field Device Language in a Process Control Language Family

E. H. Bristol
The Foxboro Co.
Dept. 0304, Bldg. N05–03A
Foxboro MA, 02035
Tel. (508) 549-2019
Fax: (508) 549-4639
email: ebristol@foxboro.com

KEYWORDS

control, language, logic, Idioms, Objects, SuperVariable

ABSTRACT

Earlier papers define a general process control language capable of integrating continuous, logic and sequencing control in a single framework. Related papers show how the underlying information modeling can emphasize intent over implementation and thus greatly improve the ease of use, readability, and self-documentation of the application defined in such a language. More recent papers show how this same technology permits the automatic construction of human interfacing with the controls and generally makes way for a much higher level integration with the plant operation. This paper introduces the possibility of families of application languages under such a model, not as they now exist, as separate control, logic, and sequencing languages, but as separate related languages, within the same overall information model, each differently optimized for easy expression of its application function and scale. The common model ensures that the languages are easily cross-learned even as they simplify application. Key is the preservation of full general purpose computational capability in each such language. The ideas are illustrated with a small general purpose language originally designed to support small "islands of automation" applications and field devices. This language would also lend itself to definition of phase logic under a higher level sequenced Batch language. The paper includes a brief discussion of its implementation both for run-time execution and self documentation.

INTRODUCTION

Process Control computer applications differ from other general purpose programmed activities in that they are carried out by a large variety of distinct, and often functionally restricted, configuration vehicles or languages. This contrasts with the power of even the smallest general purpose computer application language or the visions of powerful artificial intelligence. In part this is a result of a belief that control configuration is easier when carried out from large standard modules and traditional configuration diagrams. In fact the inflexibilities and incompatibilities of such Process Control tools contribute many difficulties, Balkanizing our efforts. This shows up as an unnecessary (implicit or explicit) division of our application talent resources into many different classes of engineers and programmers. Our initial attempts at (not so smart) smart field devices exaggerate all of the above. These computer based "smart" devices fail to even preserve their computer's inherent flexibility, for the end user. As a result the user is always hostage to the vendor's design shortsightedness. Computational resource size is not a legitimate excuse for abdicating this capability, **given adequate self-documentation**: The first interpretive computer languages successfully implemented most of the flexibilities of their larger counterparts in 8-16K-byte interpreters.

In a series of papers the author has presented a general purpose Process Control language designed to combine ease of use, flexibility, self-documentation, and integrated treatment of all dimensions of Process Control application.[1-4] More recent papers have developed the concepts of intent based information modeling underlying the language, and its benefits in automated coordination of higher level operational ad-

juncts to the controls.[5-9] This paper extends the discussion in another dimension: showing how a single language information model can be applied to create families of languages which differ in the intended, best-fit application scale with improved ease of use for each application class, without any cost in computational flexibility or cross-learning. In this case the simpler language has all of the general computational capability of the Large Language, but far less complexity with appropriately reduced application grouping mechanisms.

The earlier papers discuss a number of mechanisms shared among the family languages, including:

- The SuperVariable[10], a free form mechanism for configuring variables in tables.
- The Footnote, which allows tables and other well structured language forms to include arbitrary computations.
- Task and Activity Brackets, defining different groups of Sequenced, Looping, Parallel, and Continuously executing, and State Driven Statements.
- Idiom Loop Statements, allowing the clear definition of a continuous control structure in terms of the component control loops and their Degree of Freedom paths. This notation allows the concise statement of control purpose in terms of standardized practices within the Degree of Freedom discipline, with order of magnitude improvement in readability over normal block diagram documentation practice.
- State Oriented Logic, which allows all logic to be expressed in terms of naturally named application States, computed in Truth Tables and used in Case structures.
- Theme Statements, which allow especially structured forms to flexibly express common sequenced and coordinated functions.

The Large Language is further characterized by organization mechanisms:

- Operations; the application is divided hierarchically into Operations each acting as an Object to include the definitions of Process Variables (and IO) and control Activities and Tasks, for some subdivision of the process.
- Pages; each Operation is divided into Pages:
  - Definitions, for tabulating the SuperVariable Process Variable definitions.
  - Procedures, for including the organized groups of computational statements which generally control the process.
  - Details, for including Footnotes and similar supporting detailed specifications of the Definitions and Procedures Pages.
  - Comments, for incorporating program related descriptions.
  - Parameters, for tabulating control parameters associated with different control Blocks and Idioms.
- Tasks and Activities, representing the tasks, to be invoked by name, and nested (un-named) activities in the control program, which operate within the definition environment of the Operation and its Pages. Tasks can be called with argument lists like conventional procedures or subroutines even though they run in real time. In the larger scale, Operations may also be called like Tasks, initiating their highest level un-named Activity, within their own environment.
- Recipes; these make special use of the Task or Operation Call and its associated argument list, with special argument list support functions, to accomplish the normal intended action of Batch Recipes.

SMALLER LANGUAGE ROLES IN A LANGUAGE FAMILY

The Large Language is intended to support the control related configuration and documentation of the large, fully automated continuous or batch plant. But there are other less ambitions applications which can equally benefit from general control language usage. Some are cataloged below, in each case defining the associated benefits from general purpose flexibility. The potential of integrating such small applications, each supported appropriately, and related into the larger application, adds frosting to the cake:

1. Smart Sensors and Actuators. The microprocessor supported field device with one or more included sensor or actuator, and communicated actuator/measurement values, can benefit from language support allowing easily read user configuration and documentation.

Value of Flexibility: Including general computing language capability greatly improves the ability to support unconstrained, user unique, conversions and support computations that come up in any application, and are blocked by the usual limited smart sensor designs.

2. Small (Single) Loop Controllers based in the sensor or actuator. These devices are likely to need sensor/actuator configuration support as above, extended to include controller functions.

Value of Flexibility: Control functions include all of the need for computational flexibility of smart sensors, but they also benefit from general computation of calculated variables in control calculations such as linearization and feedforward.

3. Small general controllers supporting "islands of automation". The structure illustrated in the prototype application was originally motivated by a canning retort application. These devices would support one or more control loops as well. Often these would be driven by some simple higher level function formerly implemented as a cut cam follower, drum sequencer or similar device. In the language these functions are implemented as Theme Statements.

Value of Flexibility: Larger control devices will use more widely varying kinds of computation. By definition, an "island" is a remote location limited to its resources at hand, which therefore needs to be as flexible as it can be.

4. Batch control within the ISA S88 model calls for recipe phase logic which invokes remote equipment controllers. This is a further example of a remote small application being invoked from a larger one, benefiting from corresponding language family members for their implementation.

Value of Flexibility: The equipment controller will need all flexibility possible to support whatever phase functions are called for from the recipe.

5. Any other application whose larger operation can be accomplished by coordinated manipulation of simpler subsystems each through manipulating a small set of subsystem states and parameters.

By way of reminder: as envisioned here, the intended flexibility is accompanied by appropriate clear self-documentation and ease of use.

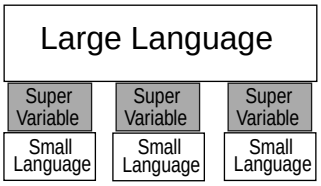
The following discussion will be based on an application falling in the third class above. But it should be recognized that there are at least the other distinct applications, which might be supported in often simpler variants of the overall language. The resulting program listing can be thought of, at least ideally, as a single page document. By comparison with the above Large Language characteristics, the Pages are replaced by six Paragraphs: Definitions, Footnotes, Conversions and Filters, Loops, Theme Statements, and Procedures. It will be seen that this simpler, restricted structure, nevertheless, imposes no computational limitations for the intended simpler application.

The critical aspect of these small applications, and the element which looms proportionately much larger in them, is the Process Variable (sensor or actuator) definition/configuration. The run-time function of the Small Language is dominated by the processing (I/O, filtering, etc.) of the associated variables. In an application in which both Large and Small Language versions coexist in coordination, the Process Variables of the Large Language would be implemented by Small Language sensors and actuators. The resulting language model envisions the Process Variable definition as the visible interface between the Small and the Large Language programs.

The SuperVariable and the Small Language

The SuperVariable represents a user level strategy for flexibly configuring the elaborate elements of the user's process variables without recourse to the typical rigid and bloated Input/Output Blocks normally associated with vendors control software. It allows the user to define sets of variables, structured according to his desires in terms of the Attributes (names, value types, scalings, engineering units, alarms, conversions, and filters) that he wants, leaving off any I/O variable functionality that he doesn't need. The Attribute types define headings in tables of the associated variables. Figure 1 represents the listing of the small application, described later. The bracketed Explicit SuperVariable section shows the typical SuperVariable specification form. It includes eleven variables arranged in six distinct tables each of a different user defined type.

The Attributes in the table are laid out for clearest documentation, but also in an order that roughly corresponds to their natural use in the normal I/O scan. Figure 1 also illustrates another Language concept especially associated with the SuperVariable: the Footnote. The numbered asterisk in the third column (the Input Attribute) of the T104 variable represents a call to execute (at the associated time in the I/O scan) the T104=(T101+T102)/2 computation listed lower in the Footnotes Paragraph.



As suggested above, with respect to the Large and Small Languages, the SuperVariable has another natural role: as the standard definition form in the Large Language for a process variable which might be associated with a field device in whose internal controls were run in the Small Language, it is the natural interface

between the two. The SuperVariable can be viewed as a software terminal board, whose parameter Attributes are visible and modifiable from either Language.

But the emphasis on the SuperVariable goes further: Since the Footnote gives the SuperVariable the ability to do any computation within the natural SuperVariable scan, it can be used in the computational implementation of all of the functions, however specified. In the Small Language, as prototyped, the compiler is used to convert each part of the listed control function into p-coded elements which are implemented (invisibly) as Footnotes merged at the appropriate point in the internal SuperVariable scan.

For this reason, the concept and implementation[11] of the Small Language are treated as a generalization of the SuperVariable and its processing, and translated into a long, continuously interpreted, run-time p-code or byte-code which represents a continuous string of Attributes and Footnotes in a single SuperVariable. The coding of this single implementing SuperVariable is such that the configuration, documenting, and protocol processes allow it to be interpreted as multiple variables, representing the natural application view. The program is self documenting since this internal SuperVariable can be sorted and back compiled to return the original pretty-printed listing!!

A Small Application and its Listing

One of the original applications, motivating the design of the Small Language, was the control of a Canning Retort. The example application abstracts some of the character of this application. It is dominated by the basic goal of controlling two temperatures in a coordinated profile shown in the Figure 2 recording, which resulted from running the example program on a simulated process. T101 is to be held constant for a timed starting period, heated on a ramp up to some Cooking temperature with a (Bias overshoot) over a specified Heating time, and held at the Cooking temperature for some Cooking time. At the same time, a remotely controlled T102 is to be initially controlled to the Cooking temperature and then (at the end of the Cooking time) cooled on a ramp down to a zero temperature over a Cooling interval. While T102 is cooling, T101 is to be cooled in a parabolic function of T102.

T101 is to be controlled in a cascade involving a fuel flow F101 and a valve V101. The flow is square root converted and all sensed values have associated Hi, Lo. and Deviation alarms. The valve has Hi and Lo alarms. The ramping parameters are represented as named variables, accessible within the application SuperVariable(s). Figure 1 is the Listing which self-documents the resulting program. It is divided into labeled[1] Paragraphs. Each Paragraph would be configured within its own appropriate GUI page. Conceptually the Listing is a single page, summarizing the entire program; but in actuality it may span several pages.

In any program, but particularly in a small system without large programming tools, one of the initial difficulties is remembering how everything has been named and spelled. Thus, the top line on each Listing or GUI page consists of a reminding list of all named variables. Following this is the application (and field device) name, and then the specification of user States for the application.[2] The body of the program follows, starting with the Definitions page, whose variables have been described. An additional variable T104 has been defined for monitoring, as the average of T100 and T102.

Some of the individual variable definitions support two equivalent names, a shorthand tag and a larger descriptive name. This possibility is just one consequence of the SuperVariable Attribute concept. All Attributes can be duplicated, for example, to give multiple Hi, Lo, or Deviation alarms when needed. The Language dot reference notation and inter-device protocol, referred to as Zipper Reference,[10] is able to access duplicated Attributes by several strategies, the simplest being duplication of Attribute tags (e.g. as in T101.HI.HI). In addition to the NAME Attribute already described, the example illustrates these At-

[1] Except the initial program heading and SuperVariable Definitions Paragraph.

[2] The application-wide States have not yet been programmed.

tributes types:

- IN/OUT, identifying I/O Input or Output channel address. (The Language includes DIN/DOUT discrete I/O addresses.)
- VALUE, identifying the real or analog value of a variable. (Also included are STATE/TIME/COUNTS value types.)
- MIN/MAX, defining the minimum/maximum real values for scaling purposes. (STATES Attributes similarly define the alternative States for State data.)
- UNITS, represent the engineering units for real values, and the units (and "scaling") for Time data.
- SET, defines the Control and alarm setpoints for controllers and Deviation alarms, based on variable type.
- HI/LO/DEV, defining the alarm limits.
- CONV/FILT, defining standardized Conversion or Filtering actions.

The Footnotes Paragraph has been described above. The Filters/Compensations Paragraph (not used in this example) allows the open ended listing of Filtering and Compensation functions to be applied to any variable. The application of these functions is specified like a Footnote, except designated with a numbered dagger, marking the Input, Output, or Value to be filtered or compensated.

The Loops Paragraph defines the continuous controls supporting the basic feedback control of variables whose setpoints are set by higher level functions (in this case, the Ramp Theme Statement described below). These are given in Idiom Loop Statements[2-4,8,9]. These statements define the Degree of Freedom paths connecting the primary variables to their associated valves, and constraint and intermediate controlled variables, in this case indicating the temperature loop driving the flow loop driving the valve.

As indicated above, the entire program is listed in a single "page", which must include the tuning parameters. As a consequence, consecutive numbers are assigned to the individual Idioms (functions) so that the associated parameters can be located in related SuperVariable like tables. The flexibility of the Language allows the individual Idioms to be extremely simple, permitting them to be very simply parameterized (five parameters for the PID based REGULATE Idioms shown, rather than the tens and hundreds common in typical vendor systems). The normal loops can fully be handled by Idioms; elaborations can be more generally programmed.

Theme Statements are illustrated by the Ramp Statement, defined in terms of an open ended sequence of Phrases, generally representing single linear segments of the ramping profile. The initial RAMP Phrase defines the main variable being ramped, although later RAMP Phrases can change the choice of main variable. The other ramping Phrases can be assigned an optional State name (in this case, *START*, *HEAT*, *COOK*, *COOL*), which the system can use, in displays, to indicate which Phrase is active, or force altered sequencing appropriately. Theme statements also support the appropriate use of Footnotes, to express side actions which are to be carried in coordination with the main statement. The † Phrase terminating Footnotes express an action or computation that is to be carried out at the end of the Phrase time interval. The embedded * Phrase Footnotes express an action or computation that is to be carried out continuously throughout the Phrase time interval.

The last Paragraph is the Procedures Paragraph, included to support any other activities or tasks needed which do not fit within the above more structured framework. They are expressed in the Bracketed procedural notation described in the references and capable of any mix of Sequenced, Looping, Parallel, Continuous, and State Driven Activities described there. Figure 3 illustrates the notation in a Large Language Batch Recipe example. In this case, the Bracketed statements constitute an unnamed Activity (within a Large Language Operation), like those used in the Procedures Paragraph here. An Activity executes when its containing entity executes. Since the Small Language program executes continuously, while it is running, every Activity in its Procedures Paragraph is continuously triggered to continue operating. It will execute unless (as a Sequenced or Parallel Activity) all of its Statements are completed.

However it can be preceded by a conditional State Prefix (a State name followed by a colon), which defines any application State intended to activate the Activity. In this case, the Activity will be executed only if the appropriate application State is set. By turning on (or off) this State, the application (or external Large Language application

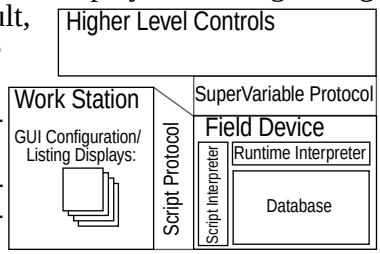
```
TOO_COLD: [COOLING, OFF
           FURNACE, ON
           FUEL, OPEN
           THERMOSTAT=80
TOO_HOT:  [FUEL, CLOSE
           FURNACE, OFF
           COOLING, OFF
           THERMOSTAT=50
```

communicating changes in State through the SuperVariable) can control the execution of the Activity.

Attributes of a Field Device Language System

The Small Language would exist in a distributed environment including the following elements:

- In a separate work station: The language editor/configurator/compiler, with GUI displays for configurating or programming the separate Paragraphs and viewing the overall listed result, which generates an abstracted script describing the implementing, run-time SuperVariable.
- A script protocol for communicating the application script from the work station to the field device.
- Optionally, in the field device or in a remote console/display: A human interface echoing the SuperVariable values and script to create an operator display fitted to the controls.[8]



- A SuperVariable communication protocol for coordinating the necessary copies of the SuperVariable information in a larger distributed system.
- In the field device: the script interpreter, which translates the application script, on an Attribute by Attribute basis, into the internal-format, run-time control database, and the run-time interpreter which carries out the programmed controls.

An Implementation Discussion

Configuring an application program like that shown in Figure 1 would involve, for each Paragraph, a GUI display formatted to accept and list the associated statements or tables and generate any supporting displayed graphics and data helpful to the user in entering that part of the application. Compiling the results into a script form, which defines the intended internal execution and the order of resulting operations, is relatively straightforward. As a consequence, the prototyping effort bypassed compiler design with a hand compiled test script entered through an appropriate script editor. The script consists of a sequence of lines each starting with an integer p-code including any necessary argument data. The script for the example application is shown in Figure 4a with a legend for the integer p-codes in Figure 4b. The left Figure 4a column lists the script lines and the right column provides a quick explanation, giving a rough sense of the script meaning. Each p-code corresponds to a basic application primitive whose internal form can be interpreted to run the application or back compile the self-documentation. The included p-codes support:

- The representation of the SuperVariable Attributes previously discussed, dealing with input, output, conversion, display, alarming, and real, State, Time, and Counts values of all variables.[3]
- A special RENAME Attribute/p-code has to do with defining user-defined names for certain Attributes.
- Various State Prefix conditional structures discussed earlier and in the references[4] but not illustrated here.[4]
- The setting off of groups of statements in Seqeunced, Parallel, Continuous, Looping, and State Driven Activities.
- Activity termination (the End code).
- LeadLag, Deadtime, and nonlinear Function elements for dynamic Filter/Compensation action.
- The beginning and end of Idiom Loop Statements.
- Fifteen Idioms for regulating, constraining, limiting, linearizing, and multiloop coordination.
- Real and Discrete expressions and assignments as developed in the Large Language references.[5]
- Truth Table logic.
- A No Operation hole for managing the program.
- A Mark code used in debugging.

[3] It will be noticed that several of these general Attributes (e.g. UNITS and SET) are translated into alternative p-code elements (e.g. RUNITS, TUNITS and RSET, TSET, SSET, CSET) which distinguish the type of variable being processed for computational convenience.

[4] The p-codes 24-27, including Scoping (SCP) and Null (NLP) Prefixes not covered here.

[5] The Real expressions are coded in Polish form as indicated in the example.

· A Wait Theme Statement.
· Six different Phrase forms making up the RAMP Theme Statement.

In comparing the listings to the compiled p-code, it will be noticed that the listing places the basic definitions at the beginning and the higher level elements later. On the other hand, the p-code picks up in order (each sample time) the application State data, computes the RAMP Theme Statement generated set points and then carries out the process variable I/O and control actions in their optimal (for control) order.

One of the major dimensions of the design is its self-documentation. Figure 5 shows the pretty printed Back Compilation from the (running) internal database. The original listing is in PostScript. The illustrated Back Compilation outputted to ASCII archival text form for lack of other available output means. Adobe Acrobat® now provides a vehicle for allowing direct PostScript display. Notice that the pretty printing allows columns and statement Phrases to be sized to fit the included data.

Figures 6a and b are provided to show some of the detailed ramifications of the Back Compilation. They were created by simply duplicating and editing the T102 section of the SuperVariable (as listed on the left of Figure 6a) of the original script of Figure 5. In Figure 6a, the Back Compilation then recognizes three identically formatted user level variables and groups them under one set of headings as shown in Figure 6a. However in Figure 6b, the middle T102 line has had its Lo alarm Attribute removed; it no longer has the same format as the preceding or following lines. The Back Compiler now provides separate heading lines for each line.

The Back Compilation operates on a number of strategies. For listing, it recognizes the beginning of distinct variables by their pattern. But internally and for execution the system still treats the collection of variables as just one long SuperVariable and Attribute list.

Summary

Process Control has not yet embraced the possibilities of true user oriented application language. This has to do with language design for ease of use and effective documentation. Ease of use requires languages fitted both to application type and to application scale. A large application requires special support for its complexity and magnitude. These features unnecessarily obscure the simple application. Simpler language supports are adequate to the organizational character of the simpler application. At the same time basic computational capabilities useful in the large application can turn out to be equally applicable to simpler applications. The paper has shown how a single Process Control information model can be uniformly expressed on more than one application/language scale.

Bibliography

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

[2] 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.

[3] E.H. Bristol, “Not a Batch Language; A Control Language”, World Batch Forum, San Francisco, May ‘94; also ISA Transactions, Dec. ‘95.

[4] E.H. Bristol, “Redesigned State Logic for an Easier to Use Control Language”, World Batch Forum, Toronto, May 13–15, ‘95.

[5] E.H. Bristol, “Batch Object Modeling and Language”, World Batch Forum, Houston, Apr. 29-30, ‘97.

[6] E.H. Bristol, “The Need for Fully Developed Process Control Examples (Use Cases)”, World Batch Forum, Baltimore, Apr. 26-29, ‘98.

[7] E.H. Bristol, “Information Models for a Software Future We Never Know”, ISA97, Anaheim CA, Oct. 5-9, ‘97.

[8] E.H. Bristol, “Deriving the Human Interface from the Automatic Controls”, Automatic Control Conference, Philadelphia, June 24-26, ‘98.

[9] E.H. Bristol, “Intent-Based Process Control Configuration Models”, ISA99, Philadelphia, Oct. ‘99.

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

[11] E.H. Bristol, “Local Equipment Controller and Method for Operating the Same”, U.S. Patent No. 5,371,895

Larger Figures

Fig. 1. Example Local Equipment Application Program

Key Names: T101 T102 T104 F101 V101 COOKTEMP BIAS VENTTIME HEATTIME COOKTIME COOLTIME

NAME: ICL_BLOCK

States: _

NAME	NAME	IN	VALUE	MIN	MAX	UNITS	SET	HI	LO	DEV
T101	Temperature_1	ECB1	_	0.0	250.0	°F	_	_	_	10.0
T104	TemperatureAv	_*[1]	_	0.0	250.0	°F	_	_	_	10.0

NAME	NAME	OUT	VALUE	MIN	MAX	UNITS	SET	HI	LO	DEV
T102	Temperature_2	T102.SET	_	0.0	250.0	°F	_	_	_	10.0

NAME	NAME	IN	CONV	VALUE	MIN	MAX	UNITS	SET	HI	LO	DEV
F101	OutletFlow1	ECB2	SQRT	_	0.0	250.0	GPM	_	_	_	10.0

NAME	OUT	VALUE	MIN	MAX	UNITS	HI	LO
V101	ECB3	_	0	100	%	90.0	5.0

NAME	VALUE	UNITS
COOKTEMP	200	°F
BIAS	7	°F

NAME	TIME	UNITS
VENTTIME	10	MIN
HEATTIME	20	MIN
COOKTIME	80	MIN
COOLTIME	15	MIN

Explicit SuperVariable

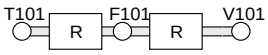
FOOTNOTES:

*[1] T104 = (T101 + T102) / 2;

FILTERS/COMPENSATIONS:

LOOPS:

T101 REGULATE[1] F101 REGULATE[2] V101;



	NAME	STATES	PB	INT	DER
[1]	T101	AUTO, +	100.0 %	0.2 MIN	0 MIN
[2]	F101	AUTO, +	100.0 %	0.1 MIN	0 MIN

THEME STATEMENTS:

RAMP T101 *START* FOR VENTTIME†[1]
HEAT TO COOKTEMP + BIAS, IN HEATTIME MIN
COOK AT COOKTEMP, FOR COOKTIME MIN
RAMP T102.VALUE *COOL* TO 0*[2], IN COOLTIME;

†[1] T102.VALUE = COOKTEMP;
*[2] T101 = COOKTEMP * (T102 / COOKTEMP)^2;

PROCEDURES:

Fig. 2. Simulated Application Response

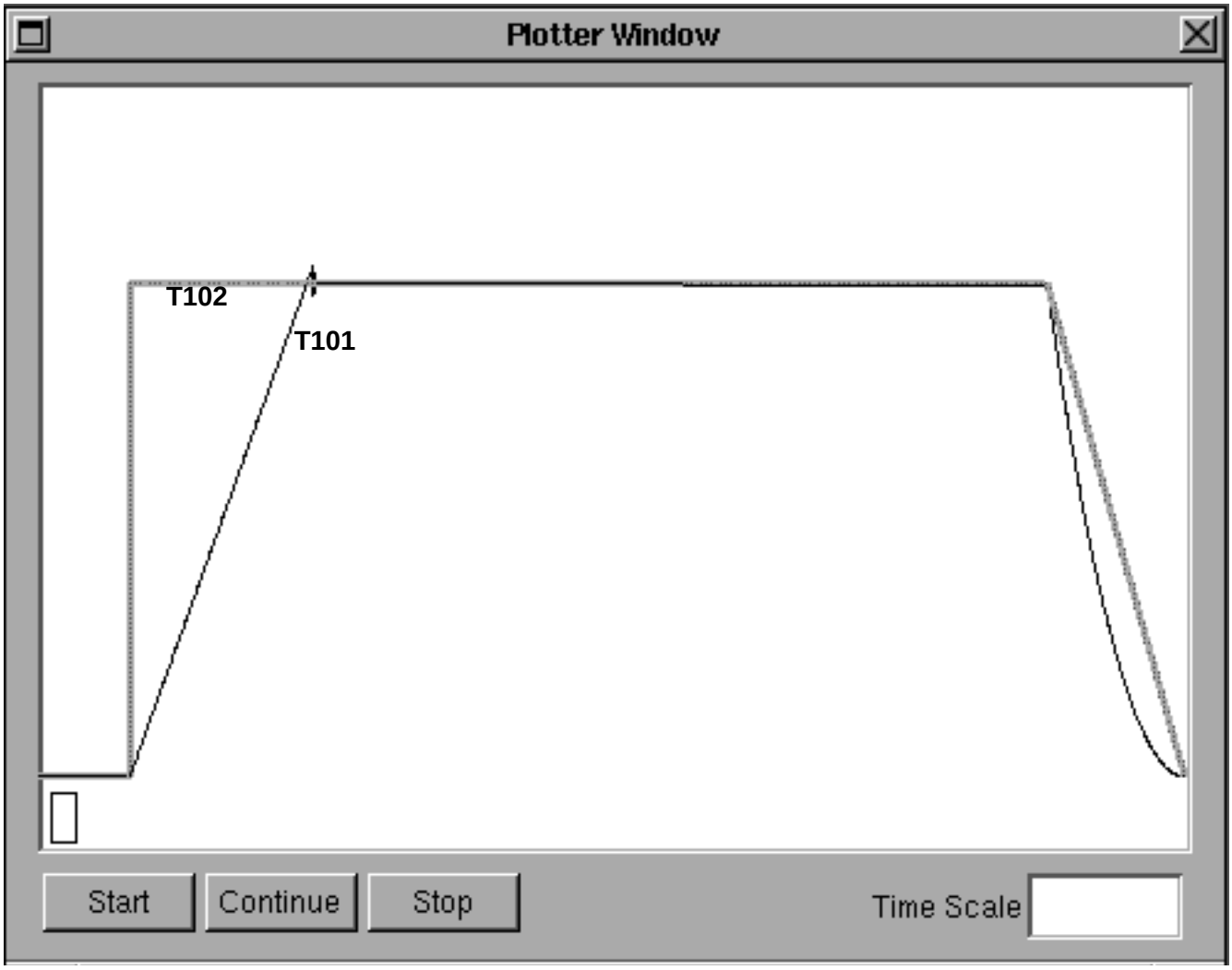


Fig. 3. Large Language Procedural Batch Recipe Example

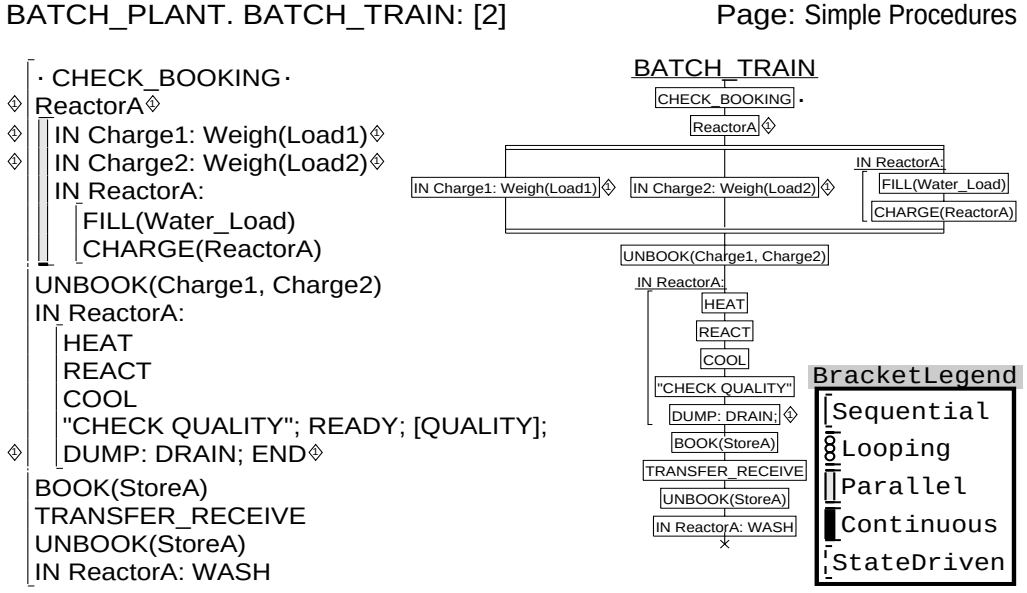


Fig. 4a. Application Script

```
1 ICL_BLOCK                 ICL_BLOCK Name pcode for Block as a whole
3                           Block State pcode, without initial State; UNDEFINED
15                          Block (User) States pcode; also UNDEFINED
62 2                        Ramp Theme Statement pcode, with first variable = T101 (Idiomatic)
64 START 2 13               Ramp_for phrase pcode, with START State name and final Footnote
                            code, VENTTIME time
54 11                       (Final) Footnote Expression pcode; COOKTEMP
55 2,3                      Footnote Assignment of Expression to T102.VALUE; Context forced (not
                            Idiomatic)
54 11 12 +                  Expression pcode (Reverse Polish) for Ramp_to_in phrase: COOK-
                            TEMP+BIAS)
66 HEAT 0 14 x              Ramp_to_in pcode, HEAT State name, no Footnote, in HEATTIME (time
                            variable first) to Expression result
65 COOK 0 15 11             Ramp_at_for pcode, COOK State name, no Footnote, in COOKTIME
                            (time variable first) at COOKTEMP
63 2,3                      Ramp2 phrase pcode; Variable becomes T102.VALUE; Context forced
                            (not Idiomatic)
66 COOL 1 16 0.0            Ramp_to_in phrase pcode, COOL State name, Continuous Footnote, in
                            COOLTIME (time variable first) to 0.0
54 11 6 11 / # 2.0 ^ *      Continuous Footnote Expression pcode (Reverse Polish): COOK-
                            TEMP*(T102/COOKTEMP)^2)
55 2                        Footnote Assignment of Expression to T101 (Setpoint; Id-
                            iomatic)
                            Blank line; End of Ramp Statement pcodes; Cannot Even
                            Have Comments or Spaces
1 T101                      Name pcode: T101
1 Temperature_1             Alternate Name pcode: Temperature_1
6 ECB1                      In pcode from ECB1
2                           Value pcode; no entry; UNDEFINED; io_state setup for IN
                            and Scaling
12 0.0 250.0                Min_Max pcode: 0.0 to 250.0 Scaling range
13 DEGF                     Real Units pcode: DEGF
16                          Real Set pcode: No value entered; UNDEFINED, default
                            0.0
37 0                        Idiom Loop pcode: Stack space for REGULATE (Initialized
                            Full Running)
39 1 0 .5 1700 0            REGULATE Idiom pcode: Loop 1, AUTO,+, PG=.5,
                            Int=170sec, Drv=0
20                          Hi Alarm pcode: No Limit entered; UNDEFINED, default 0.0
21                          Lo Alarm pcode: No Limit entered; UNDEFINED, default
                            0.0
22 10                       Dev Alarm pcode: 10.0 Deviation Limit entered
1 T104                      Name pcode: T104
1 TemperatureAv             Alternate Name pcode: TemperatureAv
6 _                         In pcode; port undefined
54 2 6 + # 2.0 /            Footnote Expression pcode (Reverse Polish):
                            (T101+T102)/2 [Value; Idiomatic]
55 4                        Footnote Assignment pcode, of Expression to T104
2                           Value pcode; no entry; UNDEFINED; io_state setup for IN
                            and Scaling
12 0 250                    Min_Max pcode: 0.0 to 250.0 Scaling range
13 DEGF                     Real Units pcode: DEGF
16                          Real Set pcode: No value entered; UNDEFINED, default
                            0.0
20                          Hi Alarm pcode: No Limit entered; UNDEFINED, default 0.0
21                          Lo Alarm pcode: No Limit entered; UNDEFINED, default
                            0.0
22 10                       Dev Alarm pcode: 10.0 Deviation Limit entered
1 T102                      Name pcode: T102
1 Temperature_2             Alternate Name pcode: Temperature_2
7 T102.SET                  Out pcode to other ICL block
2                           Value pcode; no entry; UNDEFINED; io_state setup for IN
                            and Scaling
12 0 250                    Min_Max pcode: 0.0 to 250.0 Scaling range
13 DEGF                     Real Units pcode: DEGF
16                          Real Set pcode: No value entered; UNDEFINED, default
                            0.0
20                          Hi Alarm pcode: No Limit entered; UNDEFINED, default 0.0
21                          Lo Alarm pcode: No Limit entered; UNDEFINED, default
                            0.0
22 10                       Dev Alarm pcode: 10.0 Deviation Limit entered
1 F101                      Name pcode: F101
1 OutletFlow1               Alternate Name pcode: OutletFlow1
6 ECB2                      In pcode from ECB2
11 1                        Conv pcode; Square Root Conversion
2                           Value pcode; no entry; UNDEFINED; io_state setup for IN
                            and Scaling
12 0 250                    Min_Max pcode: 0.0 to 250.0 Scaling range
13 GPM                      Real Units pcode: GPM
16                          Real Set pcode: No value entered; UNDEFINED, default
                            0.0
39 1 0 20 1000 0            REGULATE Idiom pcode: Loop 1, AUTO,+, PG=20,
                            Int=100sec, Drv=0
20                          Hi Alarm pcode: No Limit entered; UNDEFINED, default 0.0
21                          Lo Alarm pcode: No Limit entered; UNDEFINED, default
                            0.0
22 10                       Dev Alarm pcode: 10.0 Deviation Limit entered
1 V101                      Name pcode: V101
7 ECB3                      Out pcode to ECB3
2                           Value pcode; no entry; UNDEFINED; io_state setup for IN
                            and Scaling
38 1                        EndLoop pcode: Loop 1
12 0 100                    Min_Max pcode: 0.0 to 100.0 Scaling range
13 %                        Real Units pcode: %
20                          Hi Alarm pcode: No Limit entered; UNDEFINED, default 0.0
21                          Lo Alarm pcode: No Limit entered; UNDEFINED, default
                            0.0
1 COOKTEMP                  Name pcode: COOKTEMP
2 200                       Value pcode: 200.0
13 DEGF                     Real Units pcode: DEGF
1 BIAS                      Name pcode: BIAS
2 7.0                       Value pcode: 7.0
13 DEGF                     Real Units pcode: DEGF
1 VENTTIME                  Name pcode: VENTIME
4 6000                      Time pcode: 10min (600sec)
14 1                        Time Units pcode: MIN
1 HEATTIME                  Name pcode: HEATTIME
4 12000                     Time pcode: 20min (1200sec)
14 1                        Time Units pcode: MIN
1 COOKTIME                  Name pcode: COOKTIME
4 48000                     Time pcode: 80min (4800sec)
14 1                        Time Units pcode: MIN
1 COOLTIME                  Name pcode: COOLTIME
4 9000                      Time pcode: 15min (900sec)
14 1                        Time Units pcode: MIN
                            Blank line; End of Block Definition and all input pcodes; No Comments or
                            Spaces)
```

Fig. 4b. Script Legend

```
 1:  NAME          15:  STATES       29:  PARALLEL      44:  FEEDFWDX      58:  TRTB
 2:  VALUE         16:  RSET         31:  LOOPING       45:  HILIMIT       59:  HOLE
 3:  STATE         17:  SSET         32:  ST_DR         46:  LOLIMIT       60:  MARK
 4:  TIME          18:  TSET         33:  END           47:  HICONSTR      61:  WAIT
 5:  COUNTS        19:  CSET         34:  LEADLAG       48:  LOCONSTR      62:  RAMP;
 6:  IN            20:  HI           35:  DEADTIME      49:  BACKUP        63:  RAMP2
 7:  OUT           21:  LO           36:  FUNCTION      50:  SPLR          64:  RAMP_FOR;
 8:  DIN           22:  DEV          37:  IDLOOP        51:  BLEND         65:  RAMP_AT_FOR
 9:  DOUT          23:  RENAME       38:  ENDLOOP       52:  CC            66:  RAMP_TO_IN
10:  FILT          24:  LSED         39:  REGULATE      53:  MOC           67:  RMP_FRM_TO_IN
11:  CONV          25:  STP          40:  REGULATEX?    54:  REXPR
12:  MIN_MAX       26:  NLP          41:  FUNCTIONL     55:  RASSIGN
13:  RUNITS        27:  SCP          42:  FUNCTIONM     56:  DEXPR
14:  TUNITS        28:  SEQUENCE     43:  FEEDFWD       57:  DASSIGN
```

Fig. 5. Back Compiled Listing

```
<<<BLOCK>>> ICL_BLOCK
 STATES:
<<<KEY NAMES>>> T101 T104 T102 F101 V101 COOKTEMP BIAS VENTTIME HEATTIME COOKTIME COOLTIME

<<<DEFINITIONS>>>
 <<<HEADINGS>>> NAME      NAME         IN   VALUE  MIN        MAX  UNITS  SET  HI  LO   DEV
    <<<ENTRY>>> T101  TEMPERATURE_1  ECB1    _    0.000  250.000  DEGF    _   _   _   10.000
    <<<ENTRY>>> T104  TEMPERATUREAV  _*1     _    0.000  250.000  DEGF    _   _   _   10.000

 <<<HEADINGS>>> NAME      NAME         OUT     VALUE  MIN        MAX  UNITS  SET  HI  LO   DEV
    <<<ENTRY>>> T102  TEMPERATURE_2  T102.SET    _    0.000  250.000  DEGF    _   _   _   10.000

 <<<HEADINGS>>> NAME     NAME       IN   CONV  VALUE  MIN        MAX  UNITS  SET  HI  LO   DEV
    <<<ENTRY>>> F101  OUTLETFLOW1  ECB2   1      _    0.000  250.000   GPM    _   _   _   10.000

 <<<HEADINGS>>> NAME  OUT   VALUE  MIN        MAX  UNITS  HI  LO
    <<<ENTRY>>> V101  ECB3    _    0.000  100.000         _   _

 <<<HEADINGS>>>   NAME     VALUE   UNITS
    <<<ENTRY>>> COOKTEMP  200.000  DEGF
    <<<ENTRY>>>   BIAS     7.000   DEGF

 <<<HEADINGS>>>   NAME     TIME   UNITS
    <<<ENTRY>>> VENTTIME  10.000   MIN
    <<<ENTRY>>> HEATTIME  20.000   MIN
    <<<ENTRY>>> COOKTIME  80.000   MIN
    <<<ENTRY>>> COOLTIME  15.000   MIN

<<<FOOTNOTES>>>
 *1 T104 = (T101 + T102) / 2.000

<<<FILTERS/COMPENSATIONS>>>

<<<LOOPS>>>
 T101/REGULATE[1] F101/REGULATE[2] V101

 T101/           F101/           V101
     REGULATE[1]     REGULATE[2]

 <<<IDIOMH>>> STATES    PB        INT        DIR
         [1]   AUTO   200.000  2.833 MIN  0.000 MIN
         [2]   AUTO    5.000   1.667 MIN  0.000 MIN

<<<THEME_STATEMENTS>>>
  RAMP T101         START^ FOR VENTTIME †[1]
                    HEAT^  TO COOKTEMP + BIAS IN HEATTIME
                    COOK^  AT COOKTEMP FOR COOKTIME
    RAMP T102.VALUE COOL^  TO 0.000 *[2] IN COOLTIME

   †[1] T102.VALUE = COOKTEMP
   *[2] T101 = COOKTEMP * (T102 / COOKTEMP) ^ 2.000

<<<PROCEDURES>>>
```

T102

Fig. 6a. Section of the Back Compiled Listing Duplicated

```
1 T102
1 Temperature_2
7 T102.SET
2
12 0 250
13 DEGF
16
20
21
22 10
```

```
<<<HEADINGS>>> NAME      NAME         OUT     VALUE  MIN        MAX  UNITS  SET  HI  LO   DEV
   <<<ENTRY>>> T102  TEMPERATURE_2  T102.SET    _    0.000  250.000  DEGF    _   _   _   10.000
   <<<ENTRY>>> T102  TEMPERATURE_2  T102.SET    _    0.000  250.000  DEGF    _   _   _   10.000
   <<<ENTRY>>> T102  TEMPERATURE_2  T102.SET    _    0.000  250.000  DEGF    _   _   _   10.000
```

Fig. 6b. Altered Line

```
<<<HEADINGS>>> NAME      NAME         OUT     VALUE  MIN        MAX  UNITS  SET  HI  LO   DEV
   <<<ENTRY>>> T102  TEMPERATURE_2  T102.SET    _    0.000  250.000  DEGF    _   _   _   10.000

<<<HEADINGS>>> NAME      NAME         OUT     VALUE  MIN        MAX  UNITS  SET  HI   DEV
   <<<ENTRY>>> T102  TEMPERATURE_2  T102.SET    _    0.000  250.000  DEGF    _   _   10.000

<<<HEADINGS>>> NAME      NAME         OUT     VALUE  MIN        MAX  UNITS  SET  HI  LO   DEV
   <<<ENTRY>>> T102  TEMPERATURE_2  T102.SET    _    0.000  250.000  DEGF    _   _   _   10.000
```