FISHER-ROSEMOUNT

RS3™

ABC Batch Quick Reference Guide

Performance Series™ Release 1

May 1996
U.S. Manual PN: 1984-2818-1103

Fisher-Rosemount Systems, Inc.
8301 Cameron Road
Austin, Texas 78754–3895 U.S.A.

Telephone: (512) 835–2190
FAX: (512) 834–7313

FISHER·ROSEMOUNT

RS3™

ABC Batch Quick Reference Guide

About This Manual

This document is a quick reference guide to the most commonly used information about the ABC Bach Software system and Rosemount Basic Language. This document supplements the *ABC Batch Manual*. It provides immediate access to information vital to the operation of ABC Batch software.

Changes for This Release

- Added to the description of the Comm-op icon a note that provides a workaround for problems caused by using Goto and Label icons to loop around a Comm-op icon.
- Removed all references to SRU, which is now obsolete.

Revision Level for This Manual

For This Software Version:	Refer to This Document:		
	Title	Date	Part Number
P1	ABC Batch Quick Reference Guide	May 1996	1984-2818-11x3
18R2	ABC Batch Quick Reference Guide	January 1995	1984-2818-11x2
18R1	ABC Batch Quick Reference Guide	October 1993	1984-2818-11x1
15	Batch Quick Reference Guide	March 1991	1984-2814-09x2

References to Other Manuals

References to other RS3 user manuals list the manual, chapter, and sometimes the section as shown below.

Example Entries:

For ..., see CC: 3.

↑ ↑

Manual Title Chapter

For ..., see CC: 1-1.

↑ ↑

Manual Title Chapter-Section

Abbreviations of Manual Titles

AL = Alarm Messages

BA = ABC Batch

CB = ControlBlock Configuration

CC = Console Configuration

DT = Disk and Tape Functions

IO = I/O Block Configuration

OP = Operator's Guide

OV = System Overview and Glossary

PW = PeerWay Interfaces

RB = Rosemount Basic Language

RI = RNI Release Notes and Installation Guide

RP = RNI Programmer's Reference Manual

SP = Site Preparation and Installation

SV = Service

Reference Documents

Prerequisite Documents

You should be familiar with the information in the following documents before using this manual:

System Overview Manual and Glossary	1984-2640-21x0
Software Release Notes, Performance Series 1	1984-2818-0110

Related Documents

You may find the following documents helpful when using this manual:

ABC Batch Software Manual	1984-2654-21x0
Alarm Messages Manual	1984-2657-19x1
ABC Batch Quick Reference Guide	1984-2818-1103
Configuration Quick Reference Guide	1984-2812-0808
Console Configuration Manual	1984-2643-21x0
ControlBlock Configuration Manual	1984-2646-21x0
I/O Block Configuration Manual	1984-2645-21x0
Operator's Guide	1984-2647-19x1
PeerWay Interfaces Manual	1984-2650-21x0
Rosemount Basic Language Manual	1984-2653-21x0
Service Manual, Volume 1	1984-2648-21x0
Service Manual, Volume 2	1984-2648-31x0
Service Quick Reference Guide	1984-2816-0904
Site Preparation and Installation Manual	1984-2642-21x0
Software Discrepancies for Performance Series 1	1984-2818-0311
User Manual Master Index	1984-2641-21x0

Contents

Section 1:
Configuring the ABC Batch Database

Overview

The ABC Batch Database associates batch units, operations, and materials with recipes:

- Units define batch equipment and hardware for use in batch processing. The library script defines aliases for hardware addresses and tags in the batch unit. A start script defines corrective actions for the recipe to take in response to abnormal conditions.
- Operations define process activities that recipes perform on batch units. Library scripts define the sequence of steps in an operation. For example, operations for a reactor might add raw material, heat and mix the material, and release a finished product.
- Materials define equipment for use in manufacturing or processing of specific materials. The material associates the equipment with specific operations. Materials also can include values for material properties for use in the batch process. The library script defines aliases for hardware addresses and tags for the material. The material allows the recipe to use hardware that is not defined in the batch unit for specific operations.

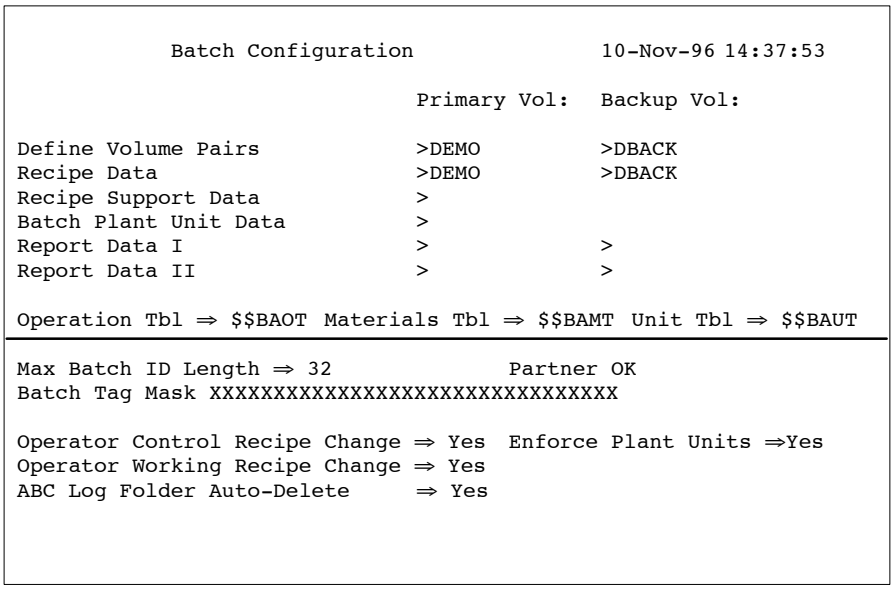
Database tables define individual entries for units, operations, and materials. The Tables are files that are stored in the ABC Data folder. In database terminology, you can think of table entries as records. The entry consists of a specific set of data fields that define the unit, operation, or material.

Some fields create links between tables or between the table and recipes. A model number links batch units to one of the 10 possible scripts that are assigned to the operation. When the recipe executes the operation, only that script with a model number that matches the batch unit is used. Operation names also link operations to materials.

Batch Configuration Screen

The Batch Configuration screen specifies:

- Disk volumes on which recipes, RBL scripts, tables, and virtual arrays are located
- Character dimensions for batch tags
- System privileges for operators
- Enforcement of plant unit ownership



Batch Configuration Screen

Rules for Redundancy

- Each time you enter primary and backup volumes in the “Define Volume Pairs” field, you enable a redundant pair.
- No two redundant volume pairs can have the same volume.
- To disable redundant pairs, reverse one of the entries: either enter the primary volume in the “Backup Vol” field or the backup volume in the “Primary Vol” field.
- If you change either the primary or backup volume on the Batch Configuration screen, you must update Master Recipes and Control Recipes that use data on these volumes.
- Redundant files require more system time to maintain.
- Working Recipes in Static mode are not updated.
- In order to start a Control Recipe from a procedure script, the Batch Run screen, procedure script, and Control Recipe must be on the same volume.
- Do not rename disk volumes that are configured as primary and backup volumes on the Batch Configuration screen.
- To delete the ABC Data folder or a file, you must delete it manually on both redundant volumes.
- By saving support data on a separate volume, you reduce system demands for space on the recipe data volume.
- By recording batch plant unit data on a separate volume, you can use a single volume as a source of batch plant unit data for all batch nodes on the PeerWay.

Configuring the Batch Configuration Screen

Procedure—Configuring the Batch Configuration Screen

Call up the Batch Configuration screen **(BAC:).**		
❑ **Define data volumes** Or ↓ Yes →	❑ **Use redundant data volumes?** **NOTE:** If you use redundant volumes, the screen automatically displays the backup volume next to the primary volume. No ↓ Yes →	Define primary and backup volumes as pairs in the “Define Volume Pairs” field. You can define as many pairs as you need, so long as no two pairs share a common volume.
	↓	← Next Procedure
	❑ **Save all recipe data on one volume (or two redundant volumes)?** No ↓ Yes →	1. Enter a primary volume for all batch data files in the ”Recipe Data” field. 2. Do not enter volumes in either the “Recipe Support Data” or “Plant Unit Data” field.
	Enter a primary volume for the following data combinations in the ”Recipe Data” field: • Recipes only • Recipes, scripts, tables and virtual arrays • Recipes and plant units	
	❑ **Save support data (scripts, tables, and virtual arrays) on a different volume than recipes?** No ↓ Yes →	Enter the primary volume for scripts, table, and virtual arrays in the “Recipe Support Data” field.

	❑ **Save batch plant unit data on a different volume than recipes?** No ↓ Yes →	Enter the primary volume for batch units in the “Plant Unit Data” field.
	❑ **Terminate redundant volume pairs?** No ↓ Yes →	1. Reverse entries for the redundant pair. Either enter the primary volume in the “Backup Vol” field or the backup volume in the “Primary Vol” field. 2. Update the configuration of all recipes that use these volumes.
↓	←	← Next Procedure
❑ **Create a tag mask?** Or ↓ Yes →	1. Enter a number from 9 to 32 in the “Max Batch ID Length” field. 2. Press [RETURN] on up to 8 Xs to select character positions in the Batch ID for the tag.	
↓	← Next Procedure	
❑ **Allow an operator to change a Control Recipe?** Or ↓ Yes →	Enter “Yes” in the “Operator Control Recipe Change” field.	
↓	← Next Procedure	
❑ **Allow an operator to change a Working Recipe?** Or ↓ Yes →	Enter “Yes” in the “Operator Working Recipe Change” field.	
↓	← Next Procedure	

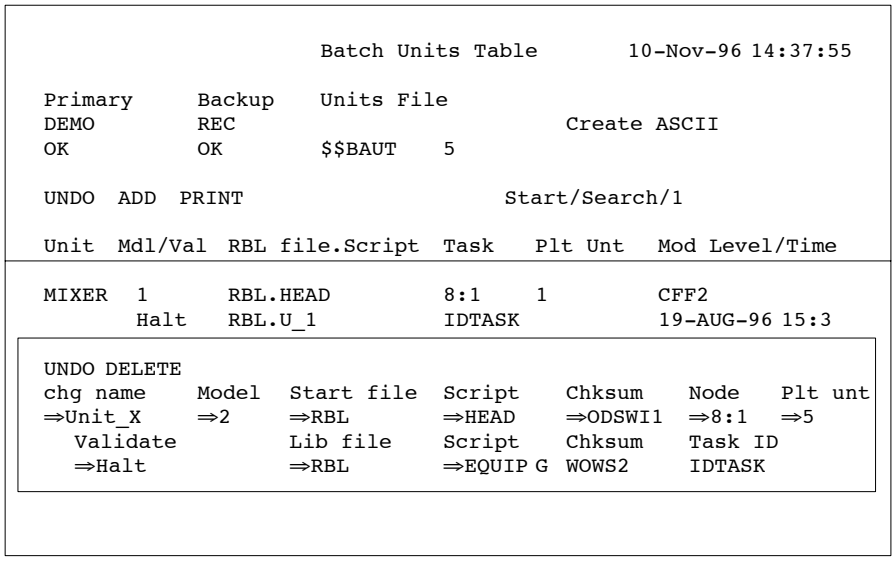
(continued on next page)

❑ **Delete the oldest Working Recipes if the ABC Log Folder is full?** **NOTE:** The oldest files are automatically deleted to make room for new files. Or ↓ Yes →	Enter "Yes" in the "ABC Log Folder Auto-Delete" field.	
↓	← Next Procedure	
❑ **Enforce Working Recipe ownership of plant units.** **NOTE:** If yes, only Working Recipes that own plant units can write to ControlBlocks that use the plant units. End↓ Yes →	1. Enter "Yes" in the "Enforce Plant Units" field. 2. For more information on: a. Working Recipe ownership of plant units, See BA: 5-2. b. RBL *ownunit* and *unownunit, see* Section 6 of this manual or RB: 1-7.	

Batch Units Table

The Batch Units Table specifies:

- The name of the batch unit
- A model number that links the batch unit with operations
- A start script that defines global *on* traps and procedures
- A library script that defines aliases for unit equipment tags and addresses
- Recipe validation actions for changes to batch units and scripts
- The address of the task that the CP uses to execute the recipe
- the tag of the Batch Run screen (use if a procedure script starts a recipe with the *begin_recipe* instruction)
- A plant unit that is available to the recipe and batch tasks



Batch Units Table

Rules for Batch Units

- DEFAULT_UN is the default name that is given to a batch unit when you create it. You must change DEFAULT_UN to another name before you can use the unit or add another unit.
- If you modify a batch unit or any of its scripts, the Control Recipe performs one of the following validation actions: None, Halt, or Warn.
- If you change the validation action on a batch unit from “None” to “Warn” or “Halt,” you must update the checksum.
- The Working Recipe can execute with batch units that do not have a library or start scripts.
- The Working Recipe cannot execute a batch unit that has a model number of 0. You can use the model number 0 to disable a batch unit.
- If the Batch Units Table cannot access the disk volume, the message “Open Failed” is displayed under the primary or backup volume name.
- If the Working Recipe changes batch units during execution, the new batch unit changes the task that runs the recipe. The batch unit specifies the task address.
- If a procedure script is used to start a Working Recipe, enter the tag of the Batch Run screen in the “Task ID” field. (The instructions *begin_recipe*, *run_recipe*, *wait_bid*, and *sendbid* are frequently used in procedure scripts.)
- If you change a script or validation action (None, Warn, or Halt), you must update the script checksum. Press [SELECT] on the checksum.

Configuring the Batch Units Table

Procedure—Configuring the Batch Units Table

Call up the Batch Units Table **(BAUT)**		
❑ **Add a unit?** Or ↓ Yes →	1. Press [SELECT] on the "Add" field to open the entry window.	
❑ **Modify a unit?** Or ↓ Yes →	1. Press [SELECT] on the unit line to open the entry window.	
	2. Enter a new name for the unit in the "Chg name" field. 3. Enter a model number between 1 and 256 in the "Model" field. 4. Enter the RBL file for the library script in the "Lib file" field. 5. Enter the RBL file for the start script in the "Start file" field. 6. Enter library and start scripts in the "Script" fields. 7. Enter the node and task number in the "Node:Task" field of the task that runs the recipe.	
	❑ **Can the recipe acquire a plant unit that is owned by another recipe?** No ↓ Yes →	Enter the plant unit number in the "Plt Unt" field. See BA: 5-2
	↓	← Next Procedure

(continued on next page)

Procedure—Configuring the Batch Units Table (continued)

	❑ **Define the validation action for the Control Recipe?** **NOTE:** In the following context, batch unit refers to the batch unit entry and its scripts. No ↓ Yes →	❑ **Ignore changes to batch units?** Yes ↓ Enter “None” in the “Validate” field. No ↓
		❑ **Warn the operator if batch units are changed?** Yes ↓ 1. Enter “Warn” in the “Validate” field. 2. If “Validate” was “None”, update the checksum. Press [ENTER] or [SELECT] on the checksum. No ↓
		❑ **Halt start-up if batch units are changed?** Yes ↓ 1. Enter “Halt” in the ”Validate” field. 2. If “Validate” was “None”, update the checksum. Press [SELECT] or [ENTER] on the checksum. No ↓
	↓	← Next Procedure
	❑ **Start a Control Recipe and/or enter a Batch ID from a procedure script (with RBL *begin_recipe*, *waitbid* and *sendbid*)?** No ↓ Yes →	Enter the tag for the Batch Run screen (BAR:) that runs the procedure script in the “Task ID” field.

(continued on next page)

Procedure—Configuring the Batch Units Table (continued)

↓	← Next Procedure	← Next Procedure
❑ **Delete a Unit?** End↓ Yes →	1. Press [SELECT] on the unit line. 2. Press [SELECT] on the “Delete” field.	

Batch Operations Table

The Batch Operations Table specifies:

- The name of the operation
- Recipe validation actions for changes to operations and scripts
- Restrictions on the operation for use with materials
- Parameter declarations for RBL variables
- Library scripts that define operations
- Script checksums for validating changes
- Model number(s) to link operations with batch units
- Labels for recipe decision icons

```
                Batch Operations Table       10-Nov-96 14:37:55

Primary   Backup    Operations File  Parameters File
DEMO      REC                                 Create ASCII
OK        OK        $$BAOT      5    OP_PARAM    8

UNDO   ADD  PRINT

Operation   Rest/Val              Scripts     Start/Search/:1
-------------------------------------------------------------
FILL        no / Halt   params    FILL_MX1 FILL_MX2 FILL_MX3
                        decision

MIX         no / Warn   params    MIX_SA1  MIX_SA2 MIX_SA3
                        decision  MIX_SA4  MIX_SA4 MIX_S55

DRAIN       no / Warn   params    DRAIN_ALL
                        decision
```

Batch Operations Table

Rules for Batch Operations

- DEFAULT_OP is the default name that is given to an operation when you create it. You must change DEFAULT_OP to another name before you can use the operation or add another operation.
- If you modify an operation or any of its scripts, the Control Recipe performs one of the following validation actions: None, Halt, or Warn.
- If the Batch Operations Table cannot access the disk volume, the message “Open Failed” is displayed under the primary or backup volume name.
- The operation name links the operation to the recipe or the operation to a batch material.
- The model number links a script that is assigned to the operation to a batch unit. If the operation is used with more than one unit, enter a range of model numbers between 1 and 256.
- The Batch Operations Table can have up to 1,200 operations.
- An operation can have up 10 scripts and 1,024 parameters.
- A script can have up to 250 of each type of variable or array that is used by parameters.
- Parameters can have four types of values: Value, String$, Time, and Boolean$. Value represents a *private local* variable; String$ represents a *private string* variable; Time represents a *private dim* array of 6 elements; and Boolean$ represents a *private local* variable.
- Parameters for all operations are stored in the Parameter File in the ABC Log folder. The Batch Operations Table can have only one parameters file at a time.

CAUTION

If you delete the Parameter File, all parameters are lost.

- In the recipe, the O–Icon operation determines branches for a subsequent Decision icon. The RBL *exit* instruction selects the decision branch.
- If you change a script or validation action (None, Warn, or Halt), you must update the script checksum. Press [SELECT] on the checksum.

Configuring the Batch Operations Table

Procedure—Configuring the Batch Operations Table

Call up the Batch Operations Table **(BADT)**.		
❑ **Add an operation?** Or ↓ Yes →	1. Press [SELECT] on the "Add" field.	
❑ **Modify an operation?** Or ↓ Yes →	1. Press [SELECT] an operation field to open a window for that field. 2. Enter a new name for the operation in the "Chg name" field.	
	❑ **Restrict use of the operation to materials?** No ↓ Yes →	Enter "No" in the field "Restrict" field. The Batch Materials Table links the operation with the material.
	↓	← Next Procedure
	❑ **Define the validation action for a Control Recipe?** **NOTE:** In the following context, operation refers to the operation entry and its scripts. No ↓ Yes →	❑ **Ignore changes to operations?** Yes ↓ Enter "None" in the "Validate" field. No ↓

(continued on next page)

Procedure—Configuring the Batch Operations Table (continued)

		❑ **Warn the user if operations are changed?** Yes ↓ 1. Enter “Warn” in the “Validate” field. 2. If “Validate” was “None”, update the checksum. Press [ENTER] or [SELECT] on the checksum. No ↓
		❑ **Halt start-up if operations are changed?** Yes ↓ 1. Enter “Halt” in the “Validate” field. 2. If “Validate” was “None”, update the checksum. Press [SELECT] or [ENTER] on the checksum. Next ↓
	↓	← Next Procedure
	❑ **Assign scripts to an operation?** No ↓ Yes →	1. To open the window, press [SELECT] on either: The “SELECT Here to enter scripts” field of a new operation, *or* An existing script if you are modifying an operation. 2. Enter from 1 to 10 RBL Files in the “File” field column. 3. Enter from 1 to 10 RBL scripts in the “Script” field column.

(continued on next page)

Procedure—Configuring the Batch Operations Table (continued)

		❑ **Do more than one batch unit models perform the operation?** Yes ↓ 4. For the script, enter a range of batch unit model numbers between 1 and 256 in the "Model Number" field that includes the model numbers for the appropriate batch units. No ↓ 5. For the script, enter the model number for a single batch unit in the "Model Number" field.
		❑ **Have you changed the validation action or modified any of the scripts?** Yes ↓ 6. Press [ENTER] or [SELECT] on the appropriate script checksums to update the checksums. No ↓
↓	←	← Next Procedure
❑ **Delete an operation?** Or ↓ Yes →	1. Press [SELECT] on the operation name to open the window. 2. Press [SELECT] on the "Delete" field.	
↓	← Next Procedure	
❑ **Define parameters for operations?** Or ↓ Yes →	Press [SELECT] on the "params" field to open the parameter window.	

(continued on next page)

Procedure—Configuring the Batch Operations Table (continued)

	❑ **Add a parameter?** No ↓ Yes →	1. Press [SELECT] on the "Add" field to create a DEFAULT_PARAMETER. 2. Change the parameter name DEFAULT_PARAMETER to a variable name. The variable must be in a script that is assigned to the operation.
	❑ **Modify a parameter?** No ↓ Yes →	1. Scroll a parameter to the entry line. 2. Change the parameter name for the operation to a variable name in one of its scripts.
	↓	← Next Procedure
	❑ **Enter the parameter type?** No ↓ Yes →	❑ **For a numeric Value?** Yes ↓ The parameter represents a *private local* variable. Enter "Value" in the "Type" field. No ↓
		❑ **For a character string?** Yes ↓ The parameter represents a *private string* variable. Enter "String$" in the "Type" field. No ↓

(continued on next page)

Procedure—Configuring the Batch Operations Table (continued)

		❑ **For a time value?** Yes ↓ The parameter represents a *private dim* array that is equal in size to the number of time values. For example, *private dim P_TIME(6*) accepts values for year, month, day, hour, minute, and second. Enter “Time” in the “Type” field. No ↓
		❑ **For a hex value?** Yes ↓ The parameter represents a *private local* variable that is is often used for ControlBlock input/output data. Enter “Boolean$” in the ”Type” field. No ↓
	↓	← Next Procedure
	❑ **Enter engineering units of measure.** No ↓ Yes →	**CAUTION** **An incorrect engineering unit can potentially confuse the operator who runs the recipe.**
↓	← Next Procedure	← Next Procedure

(continued on next page)

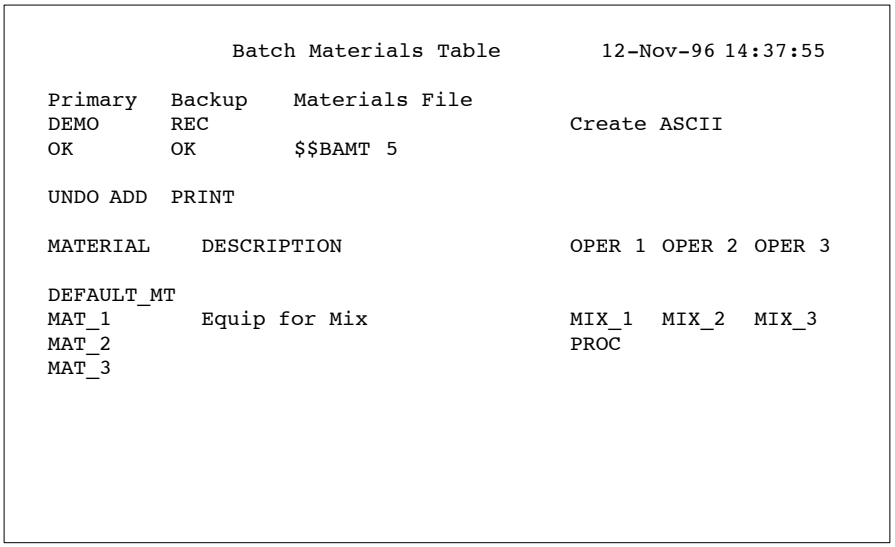
Procedure—Configuring the Batch Operations Table (continued)

❑ **Define decision labels for operations?** End↓ Yes →	❑ **Is the decision condition configured in the script?** No ↓ Yes →	**In the recipe, does the Decision icon require labels for branches?** Yes ↓ 1. Press select on the "decision" field to open the entry window. 2. Enter labels at the numbers corresponding to the branch numbers of the Decision icon. No ↓
	Configure an RBL *exit* instruction for the operation in the script that makes the decision. See the *exit* instruction in Section 6 or in RB: 1–7.	No further action is required.

Batch Materials Table

The Batch Materials Table specifies:

- Equipment that is used to manufacture or process specific materials
- Material properties
- Operations that use the material



```
              Batch Materials Table        12-Nov-96 14:37:55

Primary   Backup    Materials File
DEMO      REC                             Create ASCII
OK        OK        $$BAMT 5

UNDO ADD  PRINT

MATERIAL    DESCRIPTION                   OPER 1  OPER 2  OPER 3

DEFAULT_MT
MAT_1       Equip for Mix                 MIX_1   MIX_2   MIX_3
MAT_2                                     PROC
MAT_3
```

Batch Materials Table

Rules for Materials

- DEFAULT_MT is the default that is given to a material when you create it. You must change DEFAULT_MT to another name before you can use the material or add another material.
- Materials let you select equipment for use with specific operations so that you do not have to change the batch unit that is assigned to the recipe.
- The operation name links the material to the operation.
- A material can have links for up to 10 operations.
- In the material M-Icon in the recipe, you can select one of the 10 possible operations that use the material.
- Each material has a script that defines aliases for equipment addresses and tags for use by the operation.
- When the recipe executes a Material icon, scripts for both the material and operation are also executed.
- On the Batch Operations Table, operations must be restricted for use to the material M-Icon in the recipe.
- If you modify a material or script that is assigned to the material, the Control Recipe performs one of the following validation actions: None, Halt, or Warn.
- If you change a script or validation action (None, Warn, or Halt), you must update the script checksum. Press [SELECT] on the checksum.

Configuring the Batch Materials Table

Procedure—Configuring the Batch Materials Table

Call up the Batch Materials Table (**BAMT**).		
❑ **Add a material?** Or ↓ Yes →	1. Press [SELECT] on the "ADD" field on the Batch Materials Table. 2. Press [SELECT] on the DEFAULT_MT material in the "Material" column to open the Material Properties screen.	
❑ **Modify a material?** Or ↓ Yes →	1. Press [SELECT] on a material column to open the Material Properties screen. 2. In the Material Properties screen, press [SELECT] on the value in the "Material" field to open the window. 3. Enter a new name for the material in the "Change Name" field.	
	❑ **Define the validation action for the Control Recipe?** **NOTE:** In the following context, material refers to the material entry and its scripts. No ↓ Yes →	❑ **Ignore changes to materials?** Yes ↓ Enter "None" in the "Validate" field. No ↓

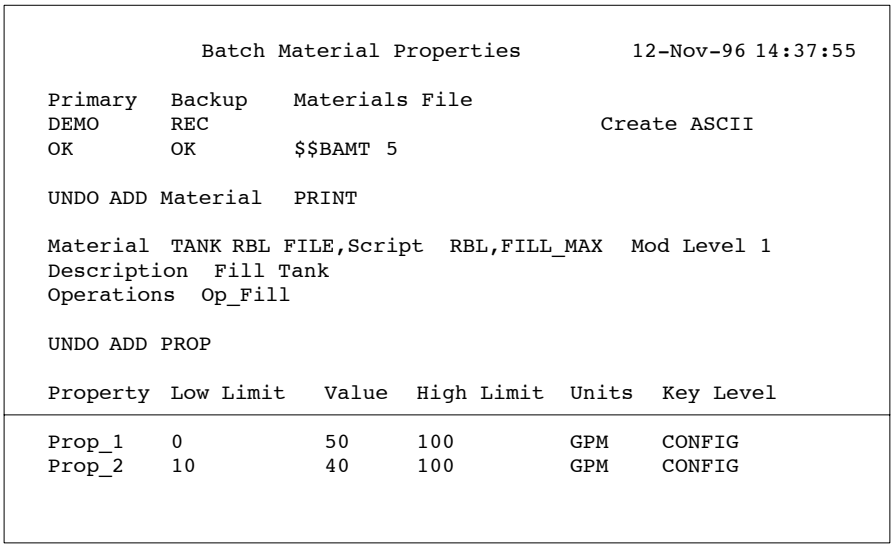
(continued on next page)

Procedure—Configuring the Batch Materials Table (continued)

		❑ **Warn the operator if materials are changed?** Yes ↓ 1. Enter “Warn” in the “Validate” field. No ↓
		❑ **Halt start-up if materials are changed?** Yes ↓ 1. Enter “Halt” in the “Validate” field. Next ↓
	↓	← Next Procedure
	❑ **Assign operations to the material?** **NOTE:** Equipment aliases must be defined in the script for the material and declared in the script for the operation. If necessary, correct scripts before proceeding. No ↓ Yes →	1. In the entry window, enter the RBL file for the script in the “RBL File” field. 2. Enter the library script in the “Script” field. 3. Close the entry window. 4. Press [SELECT] on the blank line to the right of the “Operations” field to open an entry window. 5. Enter the names of up to 10 operations that use the material.
↓	← Next Procedure	← Next Procedure
❑ **Delete a Material?** End↓ Yes →	1. Press [SELECT] on the “Material” field. 2. Press [SELECT] on the “Delete” field in the window.	

Material Properties Screen

Material properties define property values for materials for use by operations. The *getmaterial*, *getmatlim*, and *putmaterial* instructions pass values for material properties between scripts for the operation and the Material Properties screen.



```
            Batch Material Properties         12-Nov-96 14:37:55

  Primary   Backup    Materials File
  DEMO      REC                                Create ASCII
  OK        OK        $$BAMT 5

  UNDO ADD Material   PRINT

  Material  TANK RBL FILE,Script  RBL,FILL_MAX  Mod Level 1
  Description  Fill Tank
  Operations  Op_Fill

  UNDO ADD PROP

  Property  Low Limit   Value  High Limit  Units   Key Level
  ---------------------------------------------------------------
  Prop_1    0           50     100         GPM     CONFIG
  Prop_2    10          40     100         GPM     CONFIG
```

Material Properties Screen

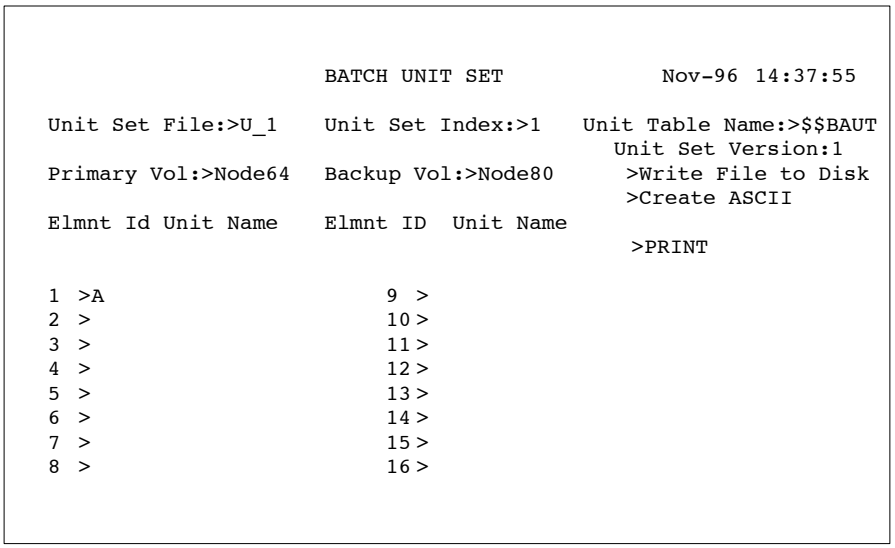
Configuring Material Properties

Procedure—Configuring Material Properties

❑ **Add a material property?** Or ↓ Yes →	1. Press [SELECT] on the "ADD PROP" field.	
❑ **Modify a material property?** Or ↓ Yes →	1. Press [SELECT] on the material property line to open the entry window.	
	2. Enter the lowest allowable property value in the "Low Limit" field. 3. Enter the highest allowable property value in the "High Limit" field. 4. Enter a property value in the "Value" field. The property value must be within the low limit and high limit values. 5. Enter the engineering unit of measure in the "Units" field. 6. Enter the user privilege type in the "Key Level" field: CNFG, RECP MGR, SUPR, or OPER.	
↓	← Next Procedure	
❑ **Delete a material?** End↓ Yes →	1. Press [SELECT] on the material property line to open the entry window. 2. Press [SELECT] on the "DELETE" field to delete the material property.	

Batch Unit Sets

A batch unit set defines an ordered set of batch units for use by the recipe. By changing batch unit sets in the recipe, you can perform the same operations on different sets of batch units. Batch unit sets are useful for multiproduct manufacturing. Each unit set can contain all the units that are used to make a particular product.



Batch Unit Set Screen

Rules for Batch Unit Sets

- The batch units that make up the batch unit set are selected from the Batch Units Table.
- A unit set file can have up to 30 batch unit sets.
- A batch unit set can have up to 16 batch units.

  NOTE: The 19-Unit Recipe limit per Main Recipe exceeds the 16-batch unit limit for batch unit sets. If you use more than 16 Unit Recipes, you cannot use a single batch unit set to make unique batch unit assignments to all Unit Recipes.
- You must assign the Unit Set File to the recipe before you can select batch unit sets. You make the assignment on the Information menu.
- Batch unit sets are assigned to the recipe either manually on the recipe or automatically by the *begin_recipe* and *mapunit* instructions in a script.
- On the recipe, you assign the batch unit set on the Save menu or Validation/Start menu.
- The element identification number is used to assign batch units in the batch unit set to Unit-Process or Start-UR icons in either the Master Recipe or Control Recipe. You make assignments on the Modify Menu.
- If you modify a batch unit in a batch unit set, you will have to reenter the element identification number in the Unit-Process icon in order to update the change in the recipe.
- During validation, the recipe uses the batch unit set version to detect modifications to the batch unit set.

Configuring Batch Unit Sets

Procedure—Configuring Batch Unit Sets

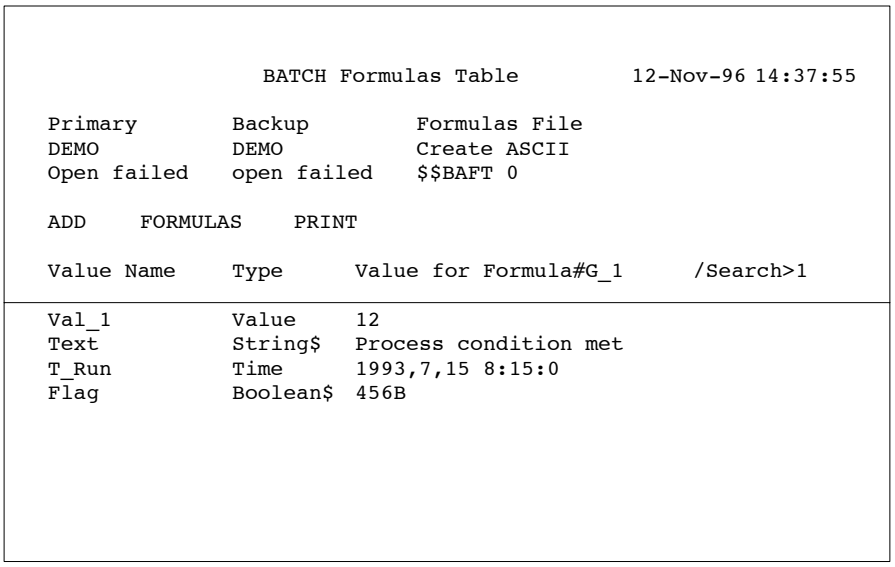
Call up a Unit Set File. **(BAUS).**		
❑ **Create a new Unit Set File?** Or ↓ Yes →	1. Enter a file name in the "Unit Set File" field. 2. After you have completed all the other entries, press [SELECT] on the "Write File to Disk" field.	
↓	← Next Procedure	
❑ **Create a batch unit set?** Or ↓ Yes →	1. Enter an index number for a batch unit set in the "Unit Set Index" field. 2. Enter the units (from the Batch Unit Table) that you want to include in the batch unit set in the "Elmnt Id Unit Name" fields. 3. Press [SELECT] on the "Write File to Disk" field.	
↓	← Next Procedure	
❑ **Change the default name of the Batch Unit Table?** Or ↓ Yes →	Enter a replacement Batch Unit Table in the "Unit Table Name" field.	
↓	← Next Procedure	

Procedure—Configuring Batch Unit Sets (continued)

❑ **Did you change the primary and backup volumes for the recipe data and/or recipe support data on the Batch Configuration screen?** End↓ Yes →	The new volumes for support data (scripts, tables, and virtual arrays) automatically appear in the "Primary Vol" and "Backup Vol" fields. You do not need to change values on the Unit Set File screen. **CAUTION** **The volume fields must match the volume fields for recipe data and/or recipe support data on the Batch Configuration screen. Otherwise, recipes will not be able to access the Unit Set File. Because these fields receive values from the Batch Configuration screen, there is no reason to change them.**	

Batch Formulas Table

A formula is a set of values that is assigned to operation parameters in a recipe. By changing formulas, you can change the values of all parameters in the recipe. Formulas are useful for multigrade processing. Each formula can contain all the parameter values necessary to make a particular grade of product.



```
                  BATCH Formulas Table         12-Nov-96 14:37:55

Primary         Backup          Formulas File
DEMO            DEMO            Create ASCII
Open failed     open failed     $$BAFT 0

ADD     FORMULAS     PRINT

Value Name      Type       Value for Formula#G_1      /Search>1
-----------------------------------------------------------------
Val_1           Value      12
Text            String$    Process condition met
T_Run           Time       1993,7,15 8:15:0
Flag            Boolean$   456B
```

Batch Formulas Table

Rules for Formulas

- You cannot assign a Formulas Table to a Control Recipe. You must assign the Formulas Table to the Master Recipe and create or overwrite the Control Recipe from the Master Recipe.
- Formulas are assigned to the recipe either manually on the Save Menu, Valid/Start Menu, or Parameters Menu or automatically with the *begin_recipe* instruction in a script.
- By changing the formula, you can change parameter values for all parameters in the recipe. Parameters are used by recipe O-Icon (operation) and M-Icon (material) types.
- On the Control Recipe, you can change parameter values manually after you assign a formula. However, values that are changed are only valid while running the recipe and cannot be saved.
- Parameters are not validated. The recipe cannot detect changes that are made to parameters in the Batch Operations Table.
- The Formulas Table can have up to 32,000 entries. Entries consist of formula names, value names, and values. Blank values do not count as entries.
- A formula can have up to 1,000 value names.
- A formula name can have up to16 alphanumeric characters.
- A value name can have up to 10 alphanumeric characters.
- If you modify a formula, the Control Recipe performs one of the following validation actions: None, Halt, or Warn.
- The recipe uses the modification time and approval time to detect changes to the formula during validation. The modification time updates automatically each time the formula is changed and will differ from the approval time, if it is not updated manually. If the modification time and approval time differ, the recipe performs the specified validation action: None, Halt, or Warn.
- If you do not want the recipe to perform the specified validation action if a formula is changed, approve the formula so that the approval time matches the modification time.
- Formula values use the same value types as parameters: Value, String$, Time, and Boolean$.
- The value name locks the formula type. After you enter a value name, you cannot change the formula type.

Configuring Formulas

Procedure—Configuring Formulas

Call up the Batch Formulas Table **(BAFT)**.		
❑ **Create a formula?** Or ↓ Yes →	Press [SELECT] on the "Formula" field to call up the formula declaration window.	
	❑ **Add a formula?** No ↓ Yes →	1. Press [SELECT] on the "Add" field to call up the formula fields. 2. Enter the formula name in the "Formula" field. 3. Enter the recipe validation action in the "Validation Status" field.
	↓	← Next Procedure
	❑ **Copy a formula?** No ↓ Yes →	1. Press [SELECT] on a formula in the list of formulas at the bottom of the screen. The formula appears in the "Formula" field. 2. Enter the name of the formula that you want to copy to in the "Copy_To" field. 3. Press [SELECT] on the "COPY_TO" field to copy the formula.
	↓	← Next Procedure
	❑ **Approve the formula?** Approval assigns a time to the formula for use in recipe validation. No ↓ Yes →	1. Enter the validation action in the "Validation Status" field. 2. Press [SELECT] on the "APPROVE" field.
	↓	← Next Procedure

(continued on next page)

Procedure—Configuring Formulas (continued)

	❑ **Create another formula?** No ↓ Yes →	Either add or copy another formula as described above.
	↓	← Next Procedure
	Close the formula declaration window.	
↓	← Next Procedure	
❑ **Edit formula values?** End↓ Yes →	Enter a formula in the "Value of Formula" field.	
	❑ **Add a new value?** No ↓ Yes →	1. Press [SELECT] on the "Add" field.
	❑ **Modify a value?** No ↓ Yes →	1. Press [SELECT] on the value name in the "Value Name" field.
		2. Enter the value type (Value, String$, Time, Boolean$, or novalue) in the "Type" field. 3. Enter the value name in the "Value Name" field. The value name locks the formula type. 4. Enter the value in the "Value" field. The entry line displays different value fields for each formula type. The "novalue" indicates that the parameter does not have a value.

(continued on next page)

Procedure—Configuring Formulas (continued)

	❑ **Compare formula values of two formulas?** No ↓ Yes →	Enter a formula in the "Compare Formula" field that you want to compare to the formula. The comparison is done for values with the same value name and type.
	↓	← Next Procedure
	Press [SELECT] on the "Close" field to add the value name to the Batch Formulas Table.	

Section 2:
Configuring Recipes

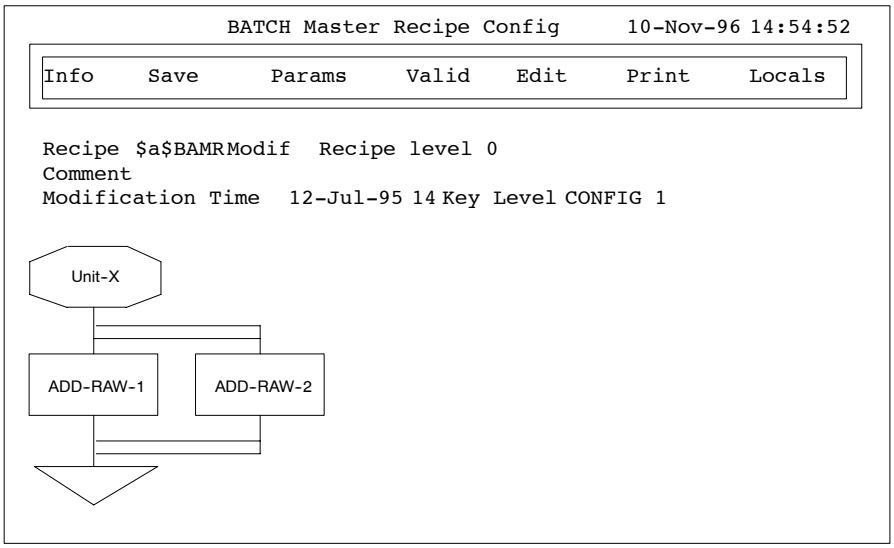
Overview

The Master Recipe defines operations and materials that are used in the batch process. The Control Recipe links operations with batch units and starts the batch process. You create the Control Recipe from the Master Recipe.

A flow chart consisting of icons graphically illustrates the sequence of operations that the recipe performs in the batch process. Each batch process begins with the batch unit that is specified in the Unit-Process icon. By executing other Unit-Process icons, the recipe can change the batch unit at various points in the batch process.

You can use Unit Recipes to execute batch units in parallel. The Main Recipe starts the Unit Recipe.

By choosing different batch units in subsequent executions of a recipe, you can perform the same operations on different batch units. By changing parameters, you can vary process values for operations. This flexibility enables you can create different products or grades of products with the same recipe.



Batch Master Recipe

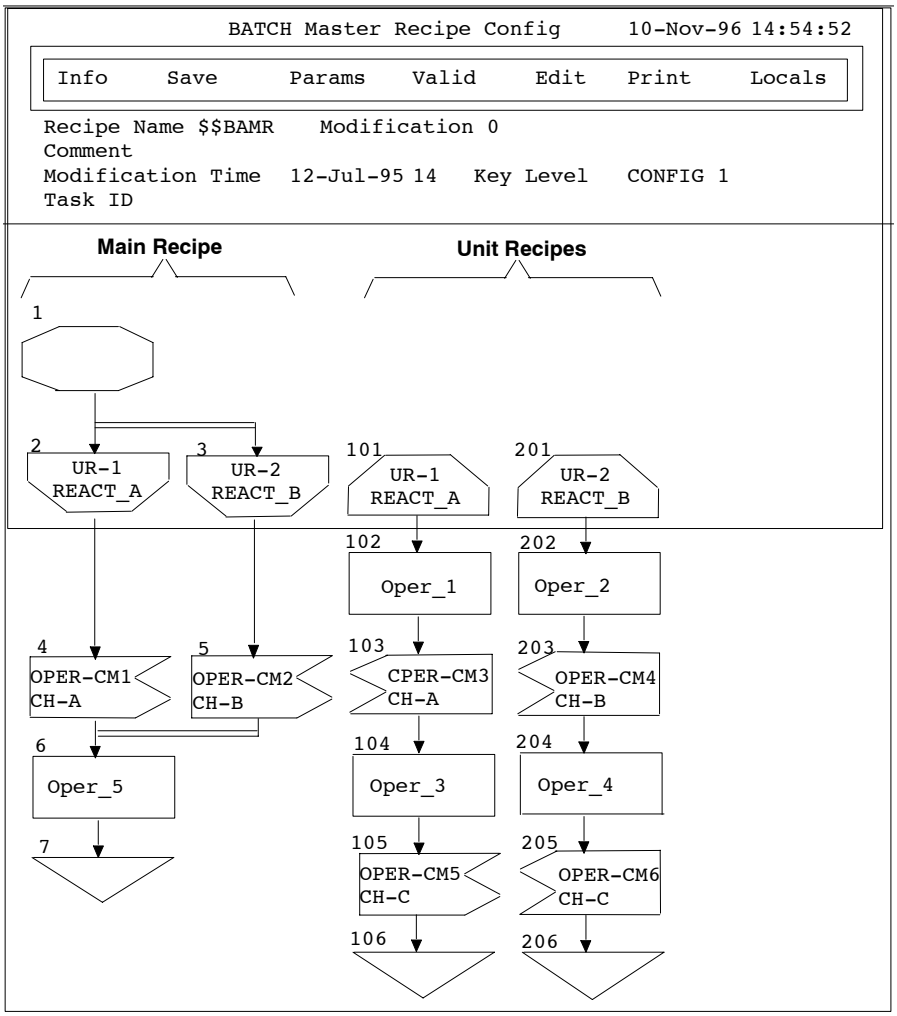
Main Recipe and Unit Recipes

Two subrecipe types, the Main Recipe and Unit Recipes, are used to run batch units in parallel from the same Working Recipe. The Main Recipe and Unit Recipes coordinate parallel processing of two or more batch units.

Main Recipe	Defines all the units in the batch process and starts execution of the Unit Recipes by executing Start_UR icons. The order of Start-UR icons in the Main Recipe determines the order of execution of Unit Recipes.
Unit Recipe	Performs operations on a batch unit that is specified in the Main Recipe in a Start-UR icon. Each Unit Recipe runs in a separate CP task, which is assigned to the batch unit in the Batch Units Table.

The Main Recipe and Unit Recipes can synchronize execution with *align_on* instructions. During execution, the Main Recipe and Unit Recipes can pass variables back and forth with the *fetch* instruction. Comm-Op icons are used in pairs to execute RBL scripts that contain these instructions.

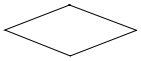
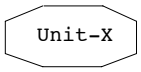
A recipe can have up to 19 Unit Recipes. However, if you exceed 16 Unit Recipes, you cannot use batch unit sets to assign batch units to all Unit Recipes. Batch unit sets are limited to 16 batch units.

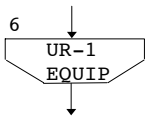
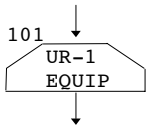
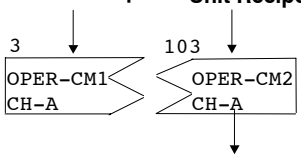
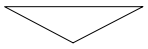


Main Recipe and Unit Recipes

Recipe Icons

Icons represent the batch units, operations, materials, and execution logic in the batch process. When linked together on the recipe, the icons form a flow chart that shows the sequence in which these components are used in the batch process.

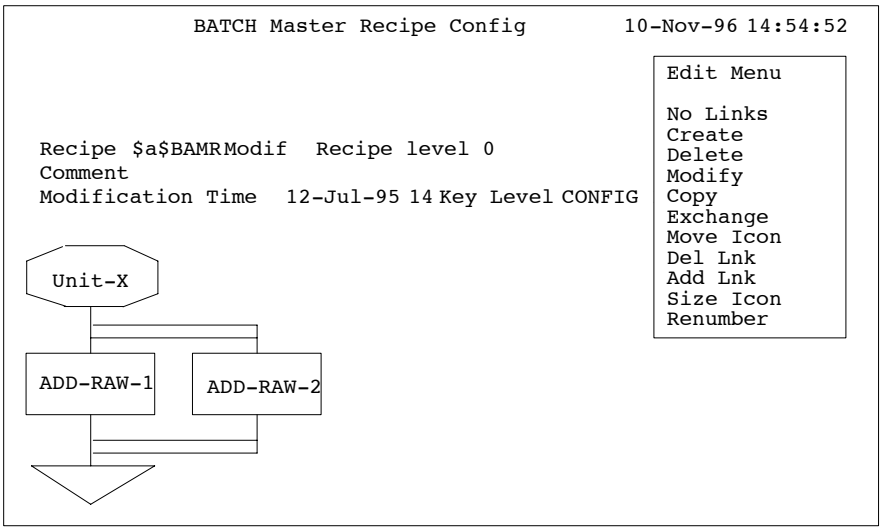
Icon	Type	Description
	O-Icon	Designates a batch operation in the batch process
	M-Icon	Designates a material and operation in which the material is used.
	Decision	Selects one of two or more paths in response to a decision in a preceding operation.
	Label	Identifies a starting point of an execution loop that is specified by a Goto icon.
	Goto	Directs execution to a Label icon in an execution loop.
	Unit-Process	Assigns or changes plant units used by the recipe.

Icon	Type	Description
Main Recipe 	Start-UR	Starts a Unit Recipe from the Main Recipe. It specifies the Unit Recipe (UR-1) and the batch unit (EQUIP) that are used by the Unit Recipe. ABC Batch uses the batch unit name as a tag name for the batch task that runs the Unit Recipe. A matching pair of Start-UR and Unit-Recipe icons links the Main Recipe and Unit Recipe.
Unit Recipe 	Unit-Recipe	Marks the start of a Unit Recipe. It specifies the name of the Unit Recipe and the batch unit that starts the Unit Recipe. The Unit-Recipe icon must have the same name as the Start-UR icon in the Main Recipe. Start-UR icons determine the order of execution of Unit Recipes.
Main Recipe or Unit Recipe **Unit Recipe** 	Comm-Op	As a pair, synchronize execution of and pass values between a Main Recipe and a Unit Recipe or two Unit Recipes. Each Comm-Op icon in the icon pair shares a common icon name and has a unique operation. Comm-Op icons use two RBL instructions to communicate: The *align_on* instruction synchronizes execution of the Comm-Op icons. The *fetch* instruction passes variables between the Comm-Op icons. **Note:** When Goto and Label icons are used to loop around a Comm-Op icon, no differentiation occurs between each instance of execution. The workaround is to modify the align_on string "alstr$" for each iteration through the loop so that the string is unique for each loop iteration, as follows: alstr$ = print$("%s.%d", "*any_string*", *loopcount)* where *loopcount* is a shared local in each looping Unit Recipe.
Main Recipe or Unit Recipe 	End_Recipe	Marks the end of the recipe.

Editing Recipes

Editing a recipe consists of several types of activities:

- Updating recipes for changes to the batch configuration.
- Resetting modification levels for changes to entries in the Batch Units Table, Operations Table, and Materials Table.
- Changing icons and links between icons on the recipe flow chart.



Edit Menu

Procedure--Editing Recipes

Call up either the Master Recipe or the Control Recipe (**BAMR** or **BACR**).		
❑ **Update the batch configuration?** Update if the changes were made on the Batch Configuration screen since the recipe was created or last updated. Or ↓ Yes →	1. Cursor to the "Info" field to call up the Info Menu. 2. Cursor to the "Config" field to call up the Config Info window. 3. Press [SELECT] on the "Update Batch Config" field to update the batch configuration for the recipe.	
↓	← Next Procedure	
❑ **Reset modification level?** Reset if changes were made to entries in the Units, Operations, or Materials Table or their scripts. Reset applies only to changes made since the recipe was created or the mod level was last reset. Or ↓ Yes →	❑ **Reset the modification level on the Master Recipe?** No ↓ Yes →	1. Cursor to the "Info" field to call up the Recipe Information menu. 2. Cursor to the "Config" field to call up the Config Info window. 3. Press [SELECT] on the "Reset Icon Mod Levels" field to reset modification levels of operations.
	↓	← Next Procedure

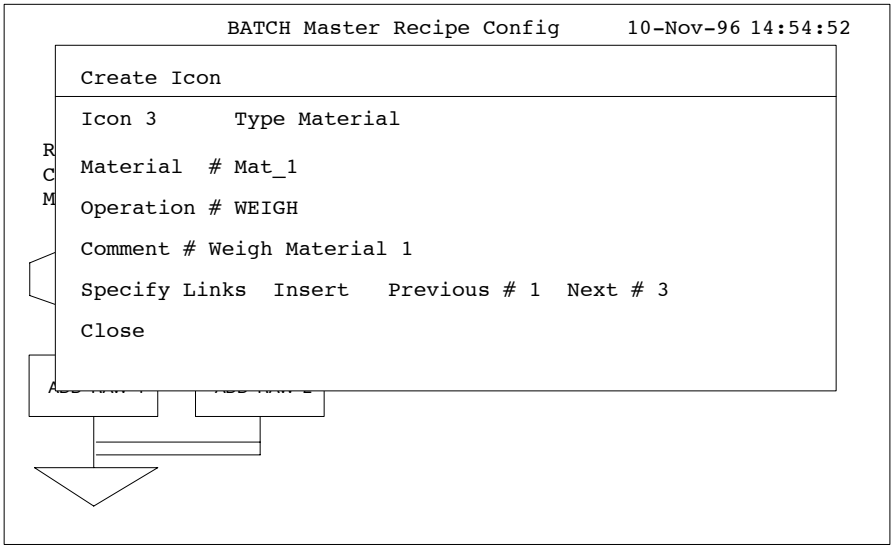
(continued on next page)

Procedure--Editing Recipes (continued)

	❑ **Reset the modification level on the Control Recipe?** No ↓ Yes →	1. Call up the original Master Recipe for the Control Recipe. 2. Reset icon modification levels as described above. 3. Press [EXCHANGE] to create a Control Recipe from the Master Recipe. 4. Cursor to the “Save” field to call up the Save Menu. 5. Enter the name of the original Control Recipe in the “Filename” field. The “Create” field should change to “Overwrite.” 6. Press [SELECT] on the “Overwrite” field.
↓	← Next Procedure	← Next Procedure
❑ **Edit the Recipe?** End↓ Yes →	Cursor to the “Edit” field on the menu bar to call up the Edit menu. For specific instructions on using edit features, see BA: 3-4.	
	❑ **Delete or modify an icon?** No ↓ Yes →	**WARNING** **If you delete and recreate an icon or change the name of an icon, the recipe updates the parameter list from the Operations Table but clears parameter values. Before you do either of these steps, make an ASCII file copy so that you can recover parameter values. Otherwise, you will have to reenter the lost values manually.**

Creating Icons

The Create Icon window is used to add icons to the Master Recipe. You cannot create icons on the Control Recipe.



Create Icon Window

Procedure—Creating Icons

Call up the Master Recipe (**BAMR**).		
1. Cursor to the “Edit” field to call up the Edit Menu. 2. Cursor to the “Create” field to call up the Create Icon window.		
❑ **Create an O-Icon for an operation?** Or ↓ Yes →	1. Enter “O-Icon” in the “Type” field. 2. Enter the operation in the “Operation” field.	
↓	← Next Procedure	
❑ **Create an M-Icon for a material?** Or ↓ Yes →	1. Enter “M-Icon” in the “Type” field. 2. Enter the material in the “Material” field. 3. Enter the operation in the “Operation” field.	
↓	← Next Procedure	
❑ **Create a Decision icon for decision branches?** Or ↓ Yes →	Enter “Decision” in the “Type” field.	
	❑ **Assign labels to decision branches?** No ↓ Yes →	1. Call up the Operations Table. 2. Press select on the “decision” field for the operation that precedes the Decision icon. 3. Enter labels at the numbers that correspond to the branch numbers of the Decision icon.
	↓	← Next Procedure

(continued on next page)

Procedure—Creating Icons (continued)

	❑ **Configure a *Decision* instruction?** No ↓ Yes →	An *exit* instruction in a script determines which branch the Decision icon selects. The script is assigned to the operation preceding the Decision icon. For information on the *exit* instruction, see Section 6 or RB: 1-7.
↓	← Next Procedure	← Next Procedure
❑ **Create a Label icon for a goto loop?** Or ↓ Yes →	1. Enter "Label" in the "Type" field. 2. Enter the name of the label in the "Identifier" field.	
↓	← Next Procedure	
❑ **Create a Goto icon for goto loop?** Or ↓ Yes →	1. Enter "Goto" in the "Type" field. 2. Enter the name of the goto icon in the "Identifier" field.	
↓	← Next Procedure	
❑ **Create a Unit-Process icon for a batch unit?** Or ↓ Yes →	Enter "Unit-Process" in the "Type" field.	
	❑ **Assign a batch unit to the Unit-Process icon when you create it?** No ↓ Yes →	1. Enter the name of the batch unit in the "Batch Unit" field. 2. Leave the "Unit Set Element" field blank.
	↓	← Next Procedure
	❑ **Assign a batch unit to the Unit-Process icon at run time?** No ↓ Yes →	1. Leave the "Batch Unit" and "Unit Set Element" fields blank. 2. See instructions for assigning batch units to recipes in this Section.

(continued on next page)

Procedure—Creating Icons (continued)

	↓	← Next Procedure
	❑ **Assign a batch unit set to all batch units in the recipe?** No ↓ Yes →	1. Leave the "Batch Unit" field blank. 2. Enter an element ID number in the "Unit Set Element" field. 3. See instructions for batch unit sets.
	↓	← Next Procedure
	❑ **Create an exit link?** No ↓ Yes →	An exit link selects an alternative path if the Unit-Process icon cannot access the batch unit. Create the link after the Unit-Process icon. The link typically connects the Unit-Process icon to a Goto icon that directs execution to another part of the recipe.
↓	← Next Procedure	← Next Procedure
❑ **Create a Start-UR icon (in the Main Recipe)?** Or ↓ Yes →	1. Enter "Start-UR" in the "Type" field. 2. Enter the name of the Unit Recipe in the "UR Name" field. **NOTE: You must create the Unit-Recipe before you can enter its name.**	
	❑ **Assign a batch unit to the Start-UR icon?** No ↓ Yes →	1. Enter an element ID number in the "Unit Set Element" field. 2. Leave the "Batch Unit" field blank.
	↓	← Next Procedure

(continued on next page)

Procedure—Creating Icons (continued)

	❑ **Assign a batch unit set to the Start-UR icon?** No ↓ Yes →	1. Enter a the batch unit in the "Batch Unit" field. 2. Leave the "Unit Set Element" field blank. 3. See instructions for batch unit sets.
↓	← Next Procedure	← Next Procedure
❑ **Create a Unit-Recipe icon (for the Unit Recipe)?** Or ↓ Yes →	1. Enter "Unit-Recipe" in the "Type" field. 2. Enter the name of the Unit Recipe in the "UR Name" field.	
↓	← Next Procedure	
❑ **Create a Comm-Op icon pair (in the Main Recipe and/or Unit Recipe)?** Or ↓ Yes →	1. Enter "Comm-Op" in the "Type" field. 2. Enter the name of the operation in the "Operation" field. 3. Enter a common name for the Comm-Op icon pair in the "Channel" field. 4. Repeat step 1 through 3 for the second Comm-Op icon in the pair.	
	❑ **Configure *align_on* and *fetch* instructions for the Comm-Op pair?** No ↓ Yes →	The *align_on* instructions synchronize execution of two Comm-Op icons. The *fetch* instruction fetches a variable from one Comm-Op operation to another. For more information on *align_on* and *fetch*, see Section 6 or RB: 1-7.
↓	← Next Procedure	← Next Procedure

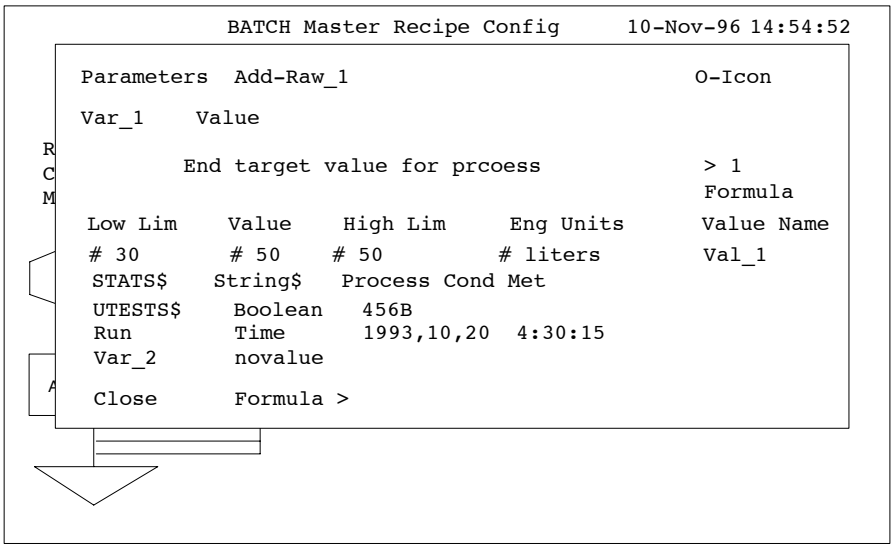
(continued on next page)

Procedure—Creating Icons (continued)

❑ **Insert the icon between two icons?** Or ↓ Yes →	1. Press [SELECT] on an icon. 2. Press [SELECT] on the “After” or “Before” field. 3. Press [SELECT] on the “Accept” field.	
↓	← Next Procedure	
❑ **Link the icon on a parallel or alternate branch?** End↓ Yes →	1. Press [SELECT] on an icon. 2. Press [SELECT] on the “Right”, “Left”, or “Link” field. The “Link” field is used to position icons on links that straddle more than one icon. 3. Press [SELECT] on the “Accept” field.	
	❑ **Is the icon a Goto or Label icon?** No ↓ Yes →	The Goto icon is always the last icon on the alternate branch and cannot have subsequent icons. Label icons on an alternate branch must have a predecessor icon. Neither Goto or Label icons are allowed in a parallel branch. For more information on illogical uses of Goto and Label icons, see BA: 3–4.

Configuring Parameters

You can use parameters to assign and change the values of process variables. Parameters are useful for creating multigrades of products from the same recipe. You must first define parameters on the Batch Operations Table.



Parameters Menu

Parameter Values and Types

Parameters can have the following types and values:

Formula	The formula for the recipe. You must first select the Formulas Table.
Value Name	The name of a formula value. The value and value type for the parameter are automatically displayed when you enter the value name.
Value	A parameter type for numeric values for *private local* variables.
Low Lim	The lowest allowable value for a parameter.
Value	The value of the parameter. If the parameter does not have a value, it displays “novalue” in the “Value” field. During validation, you can choose whether to check for parameters that do not have values.
High Lim	The highest allowable value for a parameter.
String$	A parameter type for character string values for *private string* variables.
Time	A parameter type for time values for year, month, day, hour, minute, and second for six-element *dim* arrays.
Boolean$	A parameter type for Hex values for private string variables.

Rules for Parameters

- Low limits and high limits are defined or changed on the Master Recipe. You cannot change limits on the Control Recipe.
- Value type parameters will not accept values outside the limit range.
- A change in engineering units in the Batch Operations Table causes a corresponding change in engineering units in the Parameters Menu. An incorrect engineering unit can potentially confuse the user running the recipe.
- Recipes do not automatically update parameters for changes made in the Batch Operations Table. You must update parameters in recipes. For more information on updating parameters, see BA: 3–6.
- On the Control Recipe, you can change formula values for a specific run of a Working Recipe, but you cannot save the formula values that you change.
- The Working Recipe must be suspended in Static mode in order for you to change parameter values. A Unit Recipe can be suspended in Lock mode.

Configuring Parameters

Procedure—Configuring Parameters

❑ **Create parameters for operations (on the Batch Operations Table)?** Or ↓ Yes →	See Section 1, Configuring the ABC Batch Database.	
❑ **Use formulas to assign values to parameters?** End↓ Yes →	❑ **Assign a Formulas Table to the Master Recipe?** No ↓ Yes →	1. Call up the Master Recipe. 2. Cursor to the “Info” field to call up the Information Menu. 3. Cursor to the “Recipe” field to call up the Recipe Info window. 4. Enter the Formulas Table in the “Formulas Tbl” field.
	↓	← Next Procedure

(continued on next page)

Procedure—Configuring Parameters (continued)

	❑ **Configure formulas on the Master Recipe?** No ↓ Yes →	❑ **Select the formula and value names?** Yes ↓ 1. Press [SELECT] on an O-Icon. 2. Cursor to the "Params" field to open the Parameters Menu. 3. Enter a formula in the "Formula" field. All operations in the recipe will use this formula. 4. Move a parameter to the entry line. 5. Enter a parameter value name from the formula in the "Value Name" field. The value appears in "Value" field. 6. Repeat steps 4 and 5 for additional parameters. 7. To change values for all parameters, enter another formula in the "Formulas" field. No ↓
	↓	← Next Procedure
	❑ **Configure formulas on the Control Recipe?** No ↓ Yes →	❑ **Select the formula and value names?** Yes ↓ Perform steps 1 through 7 above. No ↓

(continued on next page)

Procedure—Configuring Parameters (continued)

		❑ **Select the formula when saving the Control Recipe?** Yes ↓ 1. Cursor to the “Save” field to call up the Save menu. 2. Enter a new formula in the “Formula” field. 3. Toggle the “Check Params” field to Yes. No ↓
		❑ **Select the formula when starting the Control recipe?** Yes ↓ 1. Cursor to the “Valid/Start” field to call up the Valid/Start menu. 2. Enter the name of a new formula in the “Formula” field. 3. Toggle the “Check Params” field to Yes. No ↓
		❑ **Assign a formula with the *begin_recipe* instruction?** The *begin_recipe* instruction starts the Control Recipes from a batch script. As an option, it can assign a formula to the Control Recipe. For more information on *begin_recipe*, see Section 6 or BA: 1–7.

Updating Parameters

Recipes do not automatically update changes to parameters made in the Batch Operations Table. You can update recipes by saving the recipe to an ASCII file and then overwriting the recipe from the ASCII file.

New parameters that are updated to the recipe do not have limits or values. You cannot enter parameter limits on the Control Recipe. If you need to add or change parameter limits, either:

- Copy the Control Recipe to a remote host as an ASCII file by using ASCII File Transfer and then edit the limits on a host editor.

  or

- Edit the limits on the original Master Recipe and then overwrite the Control Recipe with the Master Recipe.

NOTE: For information on the use of ASCII file transfer to update parameters on a remote computer, see BA: 5–1.

WARNING

Two edit procedures update parameters but clear parameter values:

- **Modifying icons with the Modify window by entering the name of the operation again in the "operation" field.**
- **Deleting an icon and adding it again to the recipe.**

If you clear parameter values, you will have to reenter all parameter values again manually.

Procedure—Updating Parameters

❑ **Use this procedure if you do not have access to ASCII file transfer.** 1. Call up the Master Recipe. 2. Cursor to the “Print” field to call up the Print Menu. 3. Press [SELECT] on the “Create ASCII” or “Overwrite ASCII” field. 4. Press [SELECT] on the “Overwrite from ASCII” field to update parameters.		
❑ **Update Parameters to Control Recipe?** End↓ Yes →	1. Call up the original Master Recipe for the Control Recipe. 2. Change the parameter limits. 3. Cursor to the “Print” field to call up the Print Menu. 4. Press [SELECT] on the “Create ASCII” or “Overwrite ASCII” field. 5. Press [SELECT] on the “Overwrite from ASCII” field to update parameters. 6. Press [EXCHANGE] to call up the Control Recipe. 7. Cursor to the “Save” field to call up the Save Menu. 8. Enter the name of the original Control Recipe in the “Filename” field. The “Create” field should change to “Overwrite.” 9. Press [SELECT] on the “Overwrite” field.	

Assigning Units to Recipes

The Unit-Process icon represents a batch unit that is used by the recipe. You must assign a batch unit to the Control Recipe before you can start it. Consequently the first icon in the recipe is a Unit-Process icon. Additional Unit-Process icons and Unit Recipes can change the batch units that are used by the recipe.

You can assign units to the recipe individually or in batch unit sets. Before you can assign a batch unit set, you must select a Unit Set File. We recommend that you assign element identification numbers to Unit-Process icons before you select the batch unit set. Once you have assigned the element identification numbers, you can change all units by changing the batch unit set.

Procedure—Assigning Units to Recipes

❑ **Assign the first batch unit to the Control recipe?** Or ↓ Yes →	1. Call up the Control Recipe. 2. Cursor to the “Save” field to call up the Save Menu. 3. Enter a unit in the “BATCH Unit Name” field.	
↓	← Next Procedure	
❑ **Assign additional batch units?** Additional units change units during execution. Or ↓ Yes →	1. Call up either the Master Recipe or Control Recipe. 2. Select the Unit-Process icon or Start-UR icon (for Unit Recipes). The icon is backlighted in white. 3. Cursor to the “Edit” field to call up the Edit Menu. 4. Cursor to the “Modify” field to call up the Modify Menu. 5. Enter the unit in the “Unit Process” field.	

(continued on next page)

Procedure—Assigning Units to Recipes (continued)

❑ **Assign a batch unit set to the recipe?** The batch unit set determines the sequence of units. End↓ Yes →	❑ **Configure batch unit sets on the Master Recipe?** No ↓ Yes →	1. Call up the Master Recipe Recipe. 2. Cursor to the “Info” field to call up the Recipe Info menu. 3. Enter a Unit Set File in the “Unit Set File” field. 4. Close the Recipe Information window. 5. Select the Unit-Process or Start-UR (for Unit Recipes) icon. the icon is backlighted in white. 6. Cursor to the “Modify” field to open the Modify menu. 7. Enter an element identification number in the “Unit Set Element” field. 8. Repeat steps 6, 7, and 8 for remaining Unit-Process icons in the recipe. 9. Cursor to the “Save” field to call up the Save Menu. 10. Enter the number of the batch unit set in the “Unit Set Index” field.
	↓	← Next Procedure
	❑ **Configure the batch unit set on the Control Recipe?** No ↓ Yes →	❑ **Select the Unit Set File and units?** Yes ↓ 1. Call up the Control Recipe. 2. Perform steps 2 through 8 above. No ↓

(continued on next page)

Procedure—Assigning Units to Recipes (continued)

		❑ **Select the batch unit set when saving the Control Recipe?** Yes ↓ 1. Cursor to the “Save” field to call up the Save Menu. 2. Enter the number of the batch unit set in the “Unit Set Index” field. No ↓
		❑ **Select a batch unit set when starting the Control Recipe?** Yes ↓ 1. Cursor to the “Valid/Start” field to call up the Valid/Start Menu. 2. Enter the number of the batch unit set in the “Unit Set Index” field. No ↓
		❑ **Assign a batch unit set with the *begin_recipe* or *mapunit* instruction when starting the Control Recipe?** Yes ↓ The *begin_recipe* starts the Control Recipe from a batch script. As an option, it can assign a batch unit set to the Control Recipe at startup. The *mapunit* instruction can change the batch unit set during execution of the Working Recipe. For more information on *begin_recipe and mapunit*, see Section 6 or BA: 1–7.

Validating Recipes

When you validate the Master Recipe or the Control Recipe, the system checks the configuration for modifications to data or illogical configurations and errors that might affect execution of the Working Recipe. These include the following:

Configuration Errors in Recipes

Error	Description
Infinite Loops	A Goto icon is on a parallel branch. Goto icons can only be on alternate branches after a Decision icon.
Duplicate Label icons	You have more than one Label icon by the same name. Each Label icon must have a unique name.
Script type changed	The script type of a library or start script has been changed.
Script code changed	The script code has been changed. **NOTE:** You must update the script checksum in the Batch Units Table, Batch Operations Table, or Batch Materials Table. Press [SELECT] on the checksum.
Modification level changed	A script or table entry has been modified. The modification level for the recipe icon and the modification level table entry no longer match.
Incompatible model numbers	Model numbers for the batch unit and script for the operation do not match.
Table entry missing	A table entry has been deleted from the table or the name of the entry has been changed. Consequently, the icon can no longer reference the entry.
Material missing operation	A reference to an operation has been deleted from Batch Materials Table, or the name of the operation has been changed. Consequently, the M-Icon can no longer reference the operation.

(continued on next page)

Configuration Errors in Recipes

Error	Description
Obsolete script level	The script software level is not compatible with the ABC Batch software version. Upgrade scripts to the appropriate level.
Checksum error	The checksum for the batch script does not match the checksum that is listed for the script in the Units Table, Operations Table, or Materials Table.
Duplicate Batch ID	The Batch ID that is assigned to the Control Recipe is not unique.
Label icons used improperly	A Label icon does not have an icon that precedes it, or the Label icon is in a parallel branch.

Validating the Master Recipe

You do not have to validate the Master Recipe before creating a Control Recipe. You can validate and fix all errors to the Master Recipe on the Control Recipe.

Procedure—Validating the Master Recipe

Call up the Master Recipe. **(BAMR)**		
1. Cursor to the "Valid" field to call up the Validate Menu. 2. Press [SELECT] on the "Validate" field to validate the Master Recipe. If there are errors, the Error Log Menu appears. For explanations of error messages, see BA: 3-4.		
❑ **Are there errors in the Recipe?** Or ↓ Yes →	1. Correct the error to the recipe. 2. Validate the Master Recipe again.	
↓	← Next Procedure	
❑ **Are there errors in the database tables?** End↓ Yes →	1. Correct the errors to the effected table entries and scripts. 2. Reset the icon modification level in the Config Info window. 3. Validate the Master Recipe again.	

Validating the Control Recipe

- You can validate the Control Recipe from either the Save menu or the Valid/Start menu.
- You must assign a unique name to the Control Recipe before you can validate it.

Procedure—Validating the Control Recipe

Call up the Control Recipe (**BACR**).		
❑ **Validate and Save the Control Recipe?** Or ↓ Yes →	1. Cursor to the "Save" field to call up the Save menu. 2. If the validate option is "No," change it to "Yes." 3. To check for parameters that have no value, change the check parameters option to "Yes." 4. Press [SELECT] on the "Create" or "Overwrite" field to save the Control Recipe. The Control Recipe is validated before it is saved.	
↓	← Next Procedure	
❑ **Validate and Start the Control Recipe?** Or ↓ Yes →	1. Cursor to the "Valid/Start" field to call up the Valid/Start menu. 2. If the validate option is "No," change it to "Yes." 3. To check for parameters that have no value, change the check parameters option to "Yes." 4. Press [SELECT] on the "Start" or "Overwrite" field to start the Control Recipe. The Control Recipe is validated before it is started.	

Procedure—Validating the Control Recipe (continued)

❑ **Validate the Control Recipe only?** For explanations of error and warning messages, see BA: 3-5. Or ↓ Yes →	If you only want to validate the Control Recipe without starting or saving it, press [SELECT] on the "Validate" field on either the Save Menu or the Valid/Start menu.	
↓	← Next Procedure	
❑ **Correct Errors?** Or ↓ Yes →	❑ **Are there errors in the Recipe?** No ↓ Yes →	1. Correct the error to the recipe. 2. For information on illogical recipe configurations, see BA: 3-4.
	↓	← Next Procedure
	❑ **Are there errors in a database table or script?** No ↓ Yes →	1. Correct the errors to the effected table entries and scripts. 2. On the original Master Recipe, reset the icon modification level. 3. Save the Master Recipe. 4. Press [EXCHANGE] to call up the Control Recipe. 5. Call up the Save menu. 6. Enter the name of the original Control Recipe in the "Filename" field. The "Create" field should change to "Overwrite." 7. Validate the Control Recipe again.
↓	← Next Procedure	
❑ **Ignore or correct warnings?** End↓ Yes →	❑ **Ignore warnings?** No ↓ Yes →	On the Error Log menu, press [SELECT] on the word "WARNING." "Warning" changes to "Ignore".

Procedure—Validating the Control Recipe (continued)

	↓	← Next Procedure
	❑ **Correct warnings?** No ↓ Yes →	Follow the above steps for errors. Correct errors in the recipe or database, Reset icon modification levels on the original Master Recipe, and recreate the Control Recipe.

Saving Recipes

- Use the Save menu on the Master Recipe and Control Recipe to save the recipe.
- You must save the Master Recipe before you can create a Control Recipe.
- You must assign a unique name to the Control Recipe before you can save it.
- You can validate and start the Control Recipe without saving it.
- You can save a Control Recipe that has errors and will not validate.

Procedure—Saving the Control Recipe

Call up either the Master Recipe or the Control Recipe **(BAMR or BACR)**.		
❑ **Save the Master Recipe** Or ↓ Yes →	1. Cursor to the “Save” field to call up the Save menu. 2. Press [SELECT] on the “Save” field.	
↓	← Next Procedure	
❑ **Save the Control Recipe?** End ↓ Yes →	1. Cursor to the “Save” field to call up the Save menu. 2. Enter the Control Recipe name in the “Filename” field. 3. Set “Validate” and “Check Params” options. 4. Press [SELECT] on either the “Create” or “Overwrite” field.	

Starting a Control Recipe

- You can start the Control Recipe either manually from the Save menu or Valid/Start menu or automatically from a procedure script with a *begin_recipe* instruction.
- If you use a a *begin_recipe* instruction, you must enter the tag of the Batch Run screen for the procedure script in the “Task ID” field in the Batch Units Table.
- You must validate the Control Recipe before you can start it.
- For Batch IDs over eight characters, you must configure a tag mask on the Batch Configuration screen.

Procedure—Starting a Control Recipe

Call up the Control Recipe **(BACR)**.		
❑ **Assign batch unit(s) and a filename to the Control Recipe?** Or ↓ Yes →	For more information, see “Assigning Units to Recipes” in this section.	
↓	← Next Procedure	
❑ **Assign a Batch ID to the Control Recipe?** Or ↓ Yes →	❑ **Manually assign a Batch ID?** No ↓ Yes →	1. Cursor to the “Valid/Start” field to call up the Validation/Start menu. 2. Enter the Batch ID in the “Batch ID” field.
	↓	← Next Procedure
	❑ **Assign a Batch ID with the *begin_recipe* instruction?** No ↓ Yes →	For information, see the subsequent procedure **“Start from a batch script with the *begin_recipe* instruction.”**
	↓	← Next Procedure

(continued on next page)

Procedure—Starting a Control Recipe (continued)

	❑ **Use *waitbid* and *sendbid* instruction to assign a Batch ID?** Or ↓ Yes →	1. Call up the Batch Run screen for the script with the *waitbid* and *sendbid* instructions. 2. Press [SELECT] on the "Run" field to run the script. The *waitbid* instruction waits for the Control Recipe to start. 3. When you start the Control Recipe, the *sendbid* instruction sends a Batch ID to the Control Recipe.
↓	← Next Procedure	← Next Procedure
❑ **Start from the Save Menu?** Or ↓ Yes →	1. Validate and save the Control Recipe. 2. Cursor to the "Error Log" field to call up the Error Log menu. 3. Press [SELECT] on the "Run" field to start the Control Recipe.	
↓	← Next Procedure	
❑ **Start from the Valid/Start Menu?** No ↓ Yes →	1. Press [SELECT] on the "Start" field to validate and start the Control Recipe. 2. If there are errors or warnings, the Error Log window appears. a. Press [SELECT] on warnings to ignore them. b. Correct all errors and validate the Control Recipe again. c. Press [SELECT] on the "Run" field.	

(continued on next page)

Procedure—Starting a Control Recipe (continued)

↓	← Next Procedure	
❑ **Start from a batch script with the *begin_recipe* instruction?** End↓ Yes →	1. Validate and save the Control Recipe. 2. Call up the Batch Run screen for the script with the *begin_recip*e instruction. 3. Press [SELECT] on the "Run" field to start the Control Recipe from the batch script.	

Batch IDs and Batch Tags

When started, the Control Recipe assigns a tag to the batch task. A tag mask, which is defined on the Batch Configuration screen, creates the tag from the Batch ID. You can enter a Batch ID manually on the Control Recipe or automatically in a batch script with *waitbid* and *sendbid* instructions or the *begin_recipe* instruction.

The *begin_recipe* instruction starts the Control Recipe and assigns it a Batch ID and other values. The *waitbid* and *sendbid* instructions write the Batch ID to the Control Recipe but do not start it.

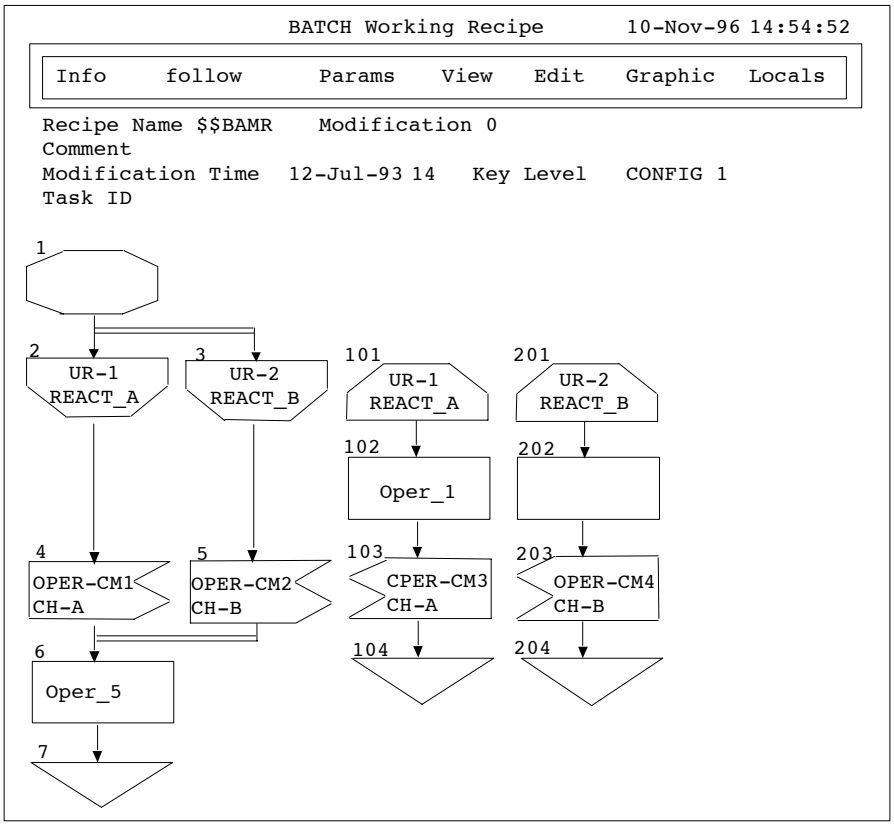
Batch IDs and batch tags have the following characteristics:

- The Batch ID and batch tag consist of alphanumeric characters. The first character in the Batch ID can be either a number or a letter. However, the first character that is selected by the tag mask must be a letter.
- A Batch ID can have up to 32 characters.
- A batch tag can have up to 8 characters. If the Batch ID is over eight characters, you must configure a tag mask on the Batch Configuration screen. Select up to 8 characters from the Batch ID.
- The Batch ID and batch tag must be unique.

Section 3:
Running the Working Recipe

Overview

The Working Recipe is used to monitor and control the execution of a batch process. The Working Recipe begins executing when you start the Control Recipe. Each time the Working Recipe runs, a record of the Working Recipe called a Finished Recipe is saved in the ABC Log folder. You can use a Finished Recipe as a record of a batch run or as a template for a Control Recipe.



Working Recipe with a Main Recipe and Unit Recipes

Working Recipe Default Colors

The Working Recipe uses the following default colors for icons:

Green	The icon has finished executing.
Yellow	The icon is executing.
Blue	The icon has not yet started executing.
Magenta	The field requires an entry or an action.
Purple	The Working Recipe skipped the icon without executing it.
Red	The icon has a runtime error.
White	The icon is highlighted when you press [SELECT] on it. Highlighting is useful for various menu activities, such as calling up the Parameters menu or tracing links.

Several other Working Recipe components have the following default colors:

Orange	The Block mode indicator.
Red	The Static mode indicator.
Yellow	Menu text.

You can change the default colors to other color selections on the Config Color Palette screen.

❑ **To call up the Config Palette screen, type:**

CCP [ENTER]

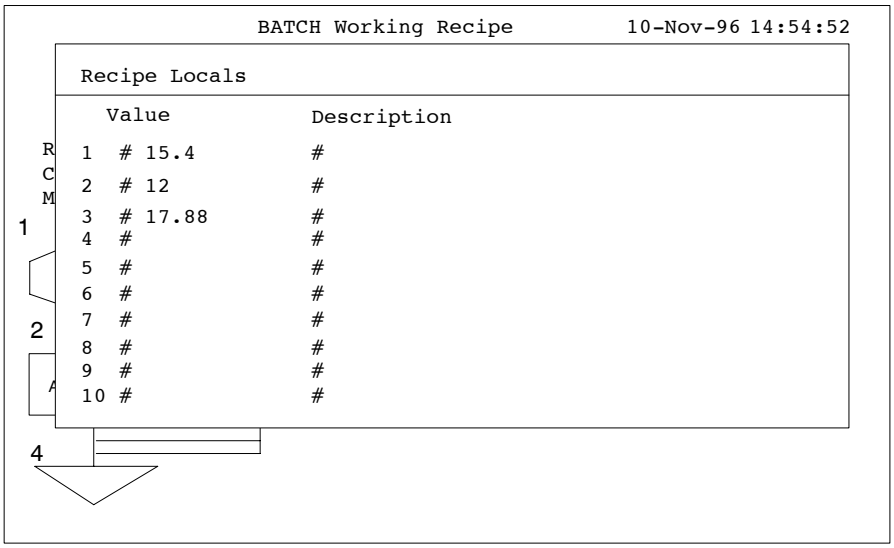
Using Recipe Locals

Recipe locals assign decimal values to variables in scripts. You can define up to 10 recipe locals in a recipe. Unlike parameters, any icon on the recipe can use the recipe local values.

Each recipe local has an identification number from 1 to 10. In the script, a *recipe_local* instruction uses the identification number to reference the value. For example:

```
recipe_local(1)
```

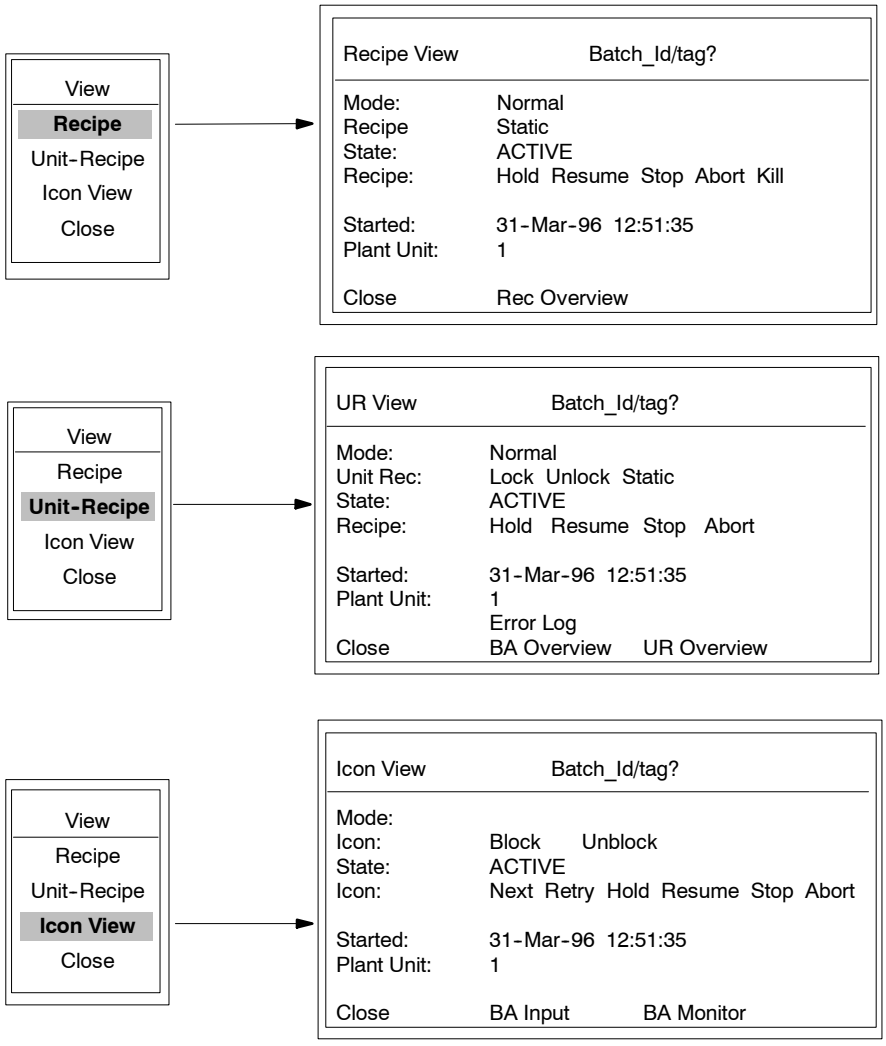
You can define recipe locals on all three recipe types: Master Recipe, Control Recipe, and Working Recipe. Only the Working Recipe can execute recipe locals. You can also use the *begin_recipe* instruction to assign recipe local values to a Control Recipe.



Recipe Locals Menu

Controlling the Working Recipe with View Screens

The Recipe View, UR View, and Icon View menus include commands that are used to control the execution of the Working Recipe, Unit Recipes, or individual icons and display status information.



View Screens

The following table describe view window fields. Some fields can apply to the recipe or to individual icons and are on more than one of the view windows.

NOTE: The following commands must be configured in the RBL scripts that are used by the operation: Abort, Cont, Halt, Hold, Next, Resume, Retry, and Stop. Otherwise, they will have no effect on recipe execution.

View Screen Fields

Abort	Interrupts the execution of the Working Recipe or operation. The "Abort" command is typically used to perform an immediate or emergency shutdown. For information on enabling the Abort command, see *on_abort* in Section 6 or RB: 1-8. For information on configuration of an abort shutdown in batch scripts, see RB: Appendix A.
BA Input	Calls up the Batch Input screen for the batch task.
BA Monitor	Calls up the Batch Monitor screen for the batch task and displays the currently executing script.
BA Overview	Calls up the Batch Overview screen.
Block	Blocks execution of subsequent icons in the same path as the executing icon. A red arrow points to the link to icons on the blocked path.
Cont	Continues a recipe suspended by a Halt command by executing an RBL *continue* instruction.
Halt	Suspends execution of the Working Recipe until you execute a "Continue" command. The Halt command is typically used to perform a gradual shutdown of the batch process when there is a prescribed sequence of activities that must be performed during the shutdown. For information on enabling the Halt command, see *on_halt* trap in Section 6 or RB: 1-8. For information on configuration of a Halt shutdown in batch scripts, see RB: Appendix A.
Hold	Suspends execution of the Working Recipe until you execute a "Resume" command. For information on enabling the Hold command, see *dishold* and *enhold* in Section 6 or RB: 1-6 and *on hold* in RB: 1-8.
Kill	Stops execution of a recipe and puts the recipe in the Idle mode.

(continued on next page)

View Screen Fields

Lock	Prevents execution of a Unit Recipe so that you can edit it without halting the rest of the recipe. To lock the Unit Recipe, position the cursor on a Unit Recipe icon and press [SELECT] on the "Lock" field.
Mode	Displays the execution mode of the recipe or icon: Normal or Static.
Next	Continues the execution of a Working Recipe that is caught in an execution loop by executing an RBL *next* instruction. For information on enabling the Next command, see *next* in Section 6 or RB: 1-6.
Plant Unit	Displays the number of the plant unit that is assigned to the Working Recipe. Plant units are specified on the Batch Units Table.
QRStart	Resumes execution of a recipe that is suspended in the Static mode without validating it.
Restart	Resumes execution of a recipe that is suspended in the Static mode. The recipe validates and saves any changes that were made while it was in Static mode. Validation errors or warnings are displayed on the Error Log menu. ❑ **To resume execution:** 1. Press [SELECT] on the "Restart" field. The recipe begins validation, and the Error Log menu appears. 2. If validation fails, acknowledge all warnings and correct the causes of all errors. After you have completed validation, the "Close" and "Run" fields appear on the Error Log menu. To discontinue validation, select the "Close" field. To resume execution, start this procedure over again. 3. Press "SELECT" on the "Run" field to start the Working Recipe. Any edit changes you made to the Working Recipe while it was in Static mode are saved. **NOTE:** If you do not want to save changes you have made, leave the Working Recipe screen and then recall it. The Working Recipe returns to its original configuration.
Resume	Resumes Working Recipe execution after a Hold command. For information on enabling the Hold command, see RB: 1-8.

(continued on next page)

View Screen Fields

Retry	Continues execution of a Working Recipe that is caught in an execution loop by executing an RBL *retry* instruction. For information on enabling the Retry command, see retry in Section 6 or RB: 1-6.
Started	Displays the date and time at which the recipe was created.
State	Displays the execution state of the recipe or icon. States includes: Active — The recipe is currently executing. Static — The recipe has suspended execution. Finished — The recipe has finished executing. Idle — The recipe is in the Idle mode and can be run again.
Static	Suspends the execution of the recipe that is in the Active mode. The Working Recipe suspends execution after the operation in progress completes execution. While the recipe is in Static mode, you can modify icons that have not been executed. The Working Recipe displays a red arrow that points to links leading to icons that have not been executed. Use the QRStart or Restart command to resume execution.
Stop	Suspends execution of a Working Recipe, as configured in the batch script. For information on enabling the Stop command, see *on stop* in Section 6 or RB: 1-8.
Unblock	Unblocks execution of blocked icons. Execution resumes at the next icon in the path.
Unlock	Unlocks execution of locked Unit Recipe icons. Execution resumes at the next icon in the Unit Recipe.

Edit Options in Active Mode

The Edit menu provides the following options in Active Mode:

Block Unblock	Blocks or unblocks the execution of an icon of an operation. Blocking an icon effectively blocks the execution of subsequent icons in the same path as the blocked icon. A red arrow points to the link leading to the blocked icon.
No Links, Next, Prev, Both	These fields turn links on and off to assist you visually in tracing links. Press [SELECT] on the field to toggle the field value. The links (before, after the icon, or both) turn red, depending on the field you select.
Size Icon	Changes the size of icons and text on the Master Recipe. Size is entered in the Size Icon window as a percentage from 1 to 100.

Edit Options in Static Mode

The Working Recipe must be in Static mode in order for you to edit it. However, you can edit a Unit Recipe that is locked, even though the Working Recipe is in Active mode.

NOTES:

- You cannot delete icons that have been run. Icons that have not been run are those that follow the arrow symbol for Static mode.
- You can copy icons that have been run and run them again.
- You cannot exchange icons that have been run with icons that have not been run.
- When you resume Active mode, the Working Recipe is validated and a copy is saved in the ABC Data folder.

The Edit menu provides the following options in Active Mode:

Add Link	Creates a link between two icons on the recipe flow chart.
Block Unblock	Blocks and Unblocks the execution of subsequent icons in the same path as the blocked icon. A red arrow points to the link leading to the blocked icon.
Copy	Copies a specified icon on the recipe flow chart on a link between two other icons.
Del Link	Deletes a link between two icons.
Delete	Deletes links to an icon on the recipe flow chart, but not the icon. The icon is moved out of the flow chart, but it is not deleted from the screen.
Exchange	Exchanges the positions of two icons on the recipe flow chart.
Modify	Changes the batch unit assigned to the Unit_Process icon.
No Links, Next, Prev, Both	These fields turn links on and off to assist you visually in tracing links. Press [SELECT] on the field to toggle the field value. The links (before, after the icon, or both) turn red, depending on the field you select.
Renumber	Renumbers icons from the chronological order in which they are created to the sequential order in which they are displayed on the screen.
Size Icon	Changes the size of icons and text on the Master Recipe. Size is entered in the Size window as a percentage from 1 to 100.

Section 4:
Task Screens

Overview

This section describes screens that are used to control and monitor batch tasks. A task is a memory partition that is used to execute one Rosemount Basic Language (RBL) script or a batch recipe. The use of tasks makes parallel processing of scripts and recipes possible in the CP or SRU. Tasks are assigned unique tags and addresses that allow the screens described in this section to identify and interface with individual tasks.

Task screens perform the following functions:

- Configuration and execution of batch tasks, using the Batch Run screen.
- Monitoring the execution of tasks on several different levels, using the Batch Monitor screen, Batch Overview screen, and Batch Acquire Queues screen.
- Reading and writing variable data between the Batch Input screen and batch task.
- Logging entries for executing tasks on the Batch Log screen.

Batch Run Screen

The Batch Run Screen allows you to configure and run scripts in a single task. It also displays available memory values.

❑ **To call up the Batch Run screen, type:**

BAR: (*node : task*) [ENTER]

❑ **To configure a task:**

1. Type tag in "Tag" field. Press [ENTER].
2. Fill in the following fields for the Unit, Formula, and Procedure scripts:

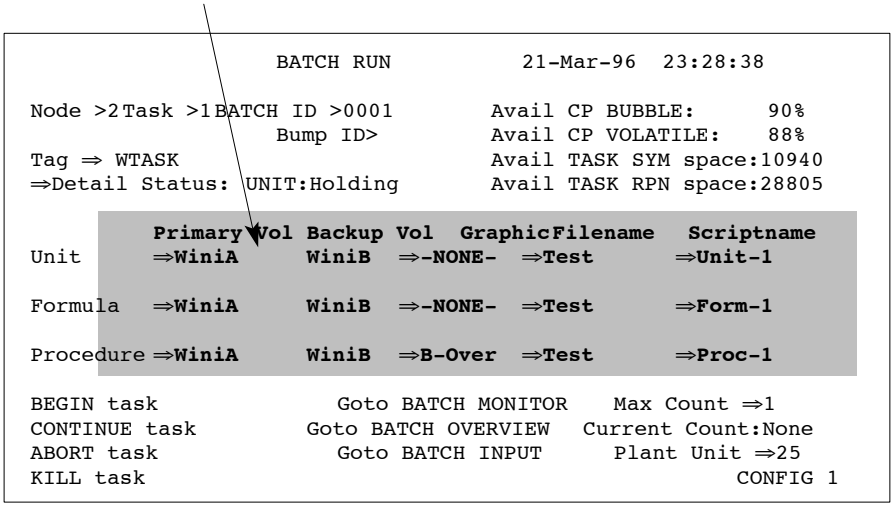
   Primary Vol on which the script resides. The backup volume is configured on the Batch Configuration screen.
   Graphic associated with the script.
   Filename of file that contains the script.
   Scriptname of script used for the task.

❑ **To call up displays associated with the configuration fields:**

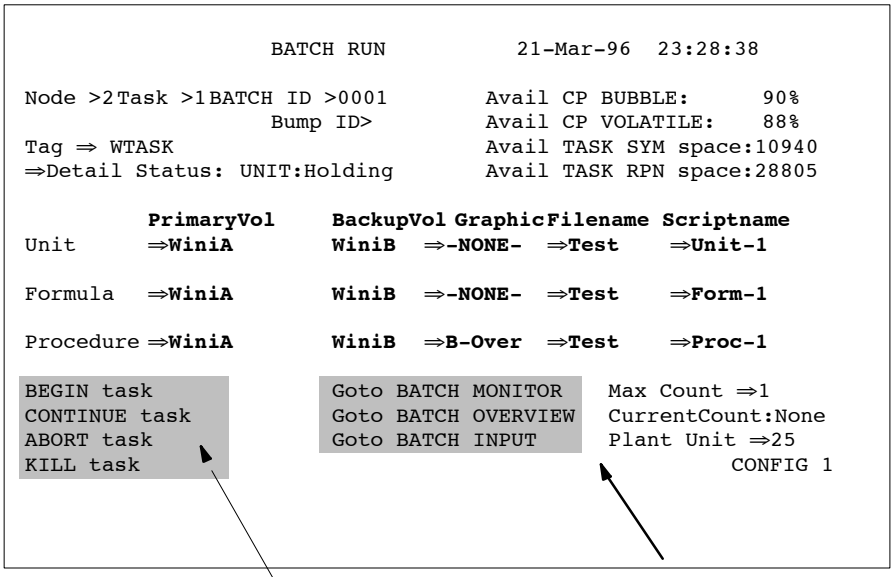
- Cursor to the field and press [SELECT].

❑ **To scroll through available configuration field options:**

- Press [NEXT OPTION] on the field.



Batch Run Screen



❑ **To Run a task:**

- Press [SELECT] or [ENTER] on the following fields:

  BEGIN task starts task that is in the idle or finished state.

  CONTINUE task continues task that is in the hold state.

  ABORT task executes an *on abort* trap. Fatal if there is no *on abort* trap in the script that is running.

  KILL task puts task in idle state.

❑ **To go to these other screens from the BATCH RUN screen:**

- Cursor to the ”Goto” field and press [SELECT].

Batch Run Screen

Batch Monitor Screen

The Batch Monitor screen displays program lines as they are being executed. It allows you to change the execution mode of the script.

❑ **To call up the Batch Monitor screen, type:**

BAM: (*node : task*) [ENTER]

❑ **To call up a Batch Script screen for the script:**

- Press [EXCHANGE].

❑ **To select script modes:**

- Cursor to one of the following operations and press [ENTER]:

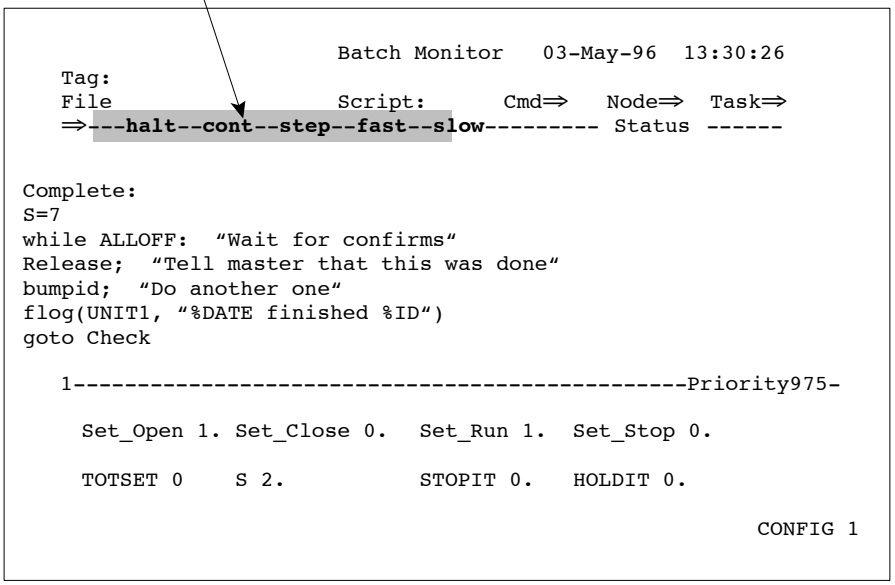
  halt stops script execution (temporary).

  cont resumes script execution after a halt or brkpt operation.

  step executes one line of a script after a “halt” or “brkpt” command.

  fast increases the priority execution speed of the script.

  slow decreases the priority execution speed of the script.



Batch Monitor Screen

❑ **To execute a script command:**

- Type or press [NEXT OPTION] to select one of the following commands in the "Cmd" field and press [ENTER]:

  none means that no command is in use.

  end terminates execution of the RBL script.

  abort executes an *on abort* trap. If there is no *on abort* trap in the script, abort is ignored.

  brkpt holds execution of the script at a line. The script mode must be in "hold". Select "brkpt", cursor to the line, and press [SELECT].

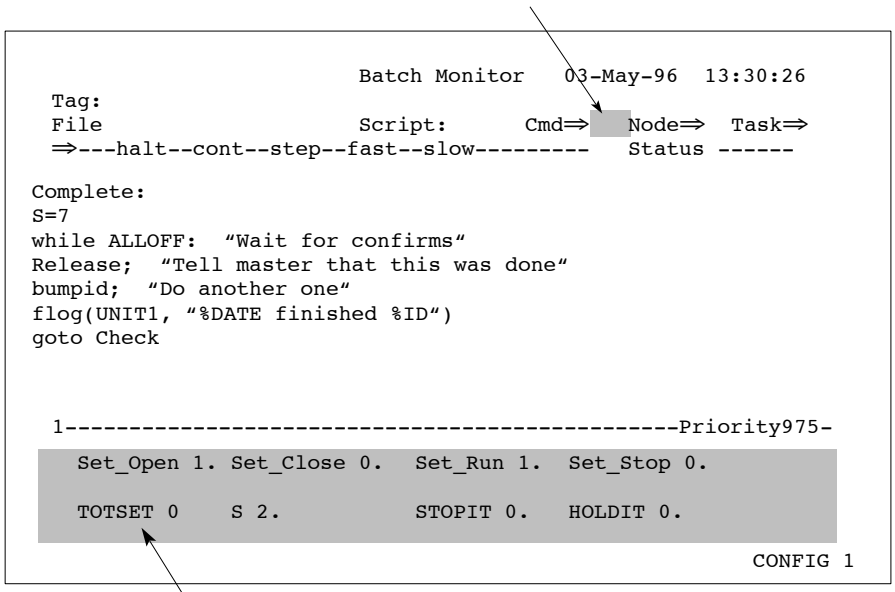
  goto directs execution of script to a line. The script mode must be in "hold". Select "goto", cursor to the line, and press [SELECT].

  break terminates a program loop.

  next terminates execution of an execution loop by executing a *next* instruction.

  retry terminates execution of an execution loop by executing a *retry* instruction.

  resume continues execution after a *hold* instruction.



❑ **To select a ControlBlock faceplate:**

- Cursor to ControlBlock tag and press [SELECT].

Batch Monitor Screen

Batch Acquire Queues Screen

The Batch Acquire Queues screen shows the actual and pending master and slave tasks that are associated with a specific task.

❑ **To call up the Batch Acquire Queues screen, type:**

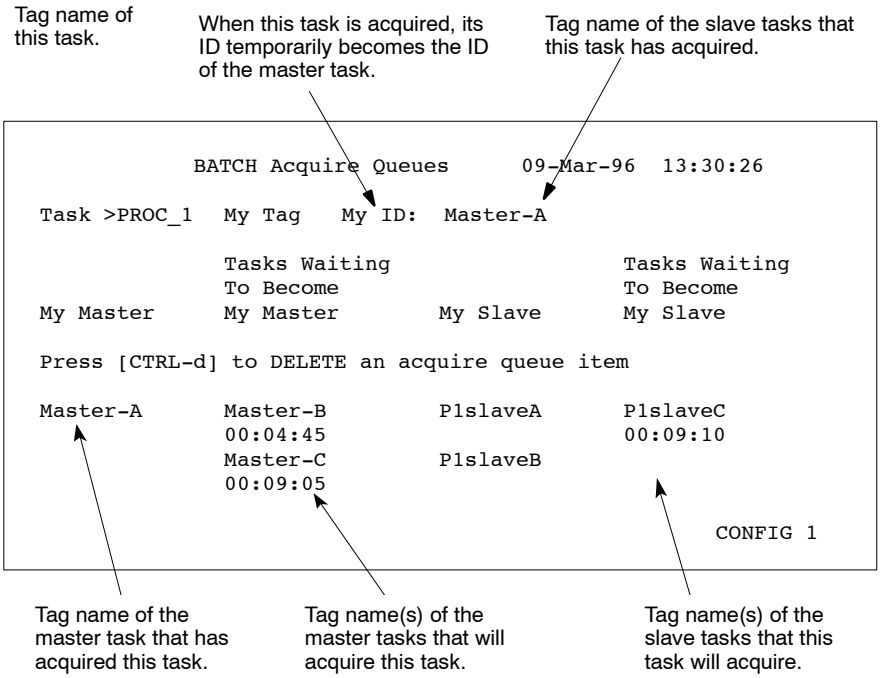
BAQ (*node* : *task*) [ENTER]

❑ **To display task information for a master or slave task:**

- Press [SELECT] on the task to call up the Batch Run screen, *or*
- Press [ENTER] on the task to call up the Batch Monitor screen.

❑ **To delete a master and slave task relationship:**

1. Put both the master and slave task in the “Hold“ mode before deleting.
2. Cursor in turn to the master and slave task and press [CTRL] [D].



Batch Acquire Queues Screen

Batch Overview Screen

Summarizes the tasks that currently reside in specified nodes. The screen displays different sets of tasks, depending upon the value in the “Node” field.

ALL TASKS	Displays all configured tasks on all nodes.
OWNED TASKS	Displays only those tasks that have a plant unit that is owned by the console.

❑ **To call up the Batch Overview screen, type:**

BAO: (*node*) [ENTER]

❑ **To select nodes, cursor to “Node” field:**

- For a specific node, type a number and press [ENTER].

or

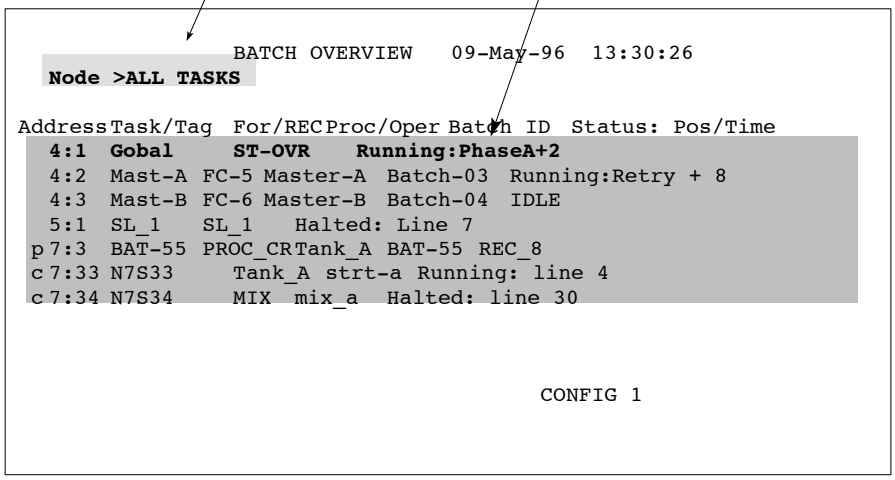
- Press [NEXT OPTION] to toggle between “ALL TASKS” and “OWNED TASKS”. Press [ENTER].

❑ **To display task information for a specified node:**

- Press [ENTER] on task to call up BATCH MONITOR screen.

or

- Press [SELECT] on task to call up BATCH RUN screen.



Batch Overview Screen

Batch Input Screen

The Batch Input screen displays user-defined messages and data requests from an active batch task. Screen prompts allow an operator to input data to the task.

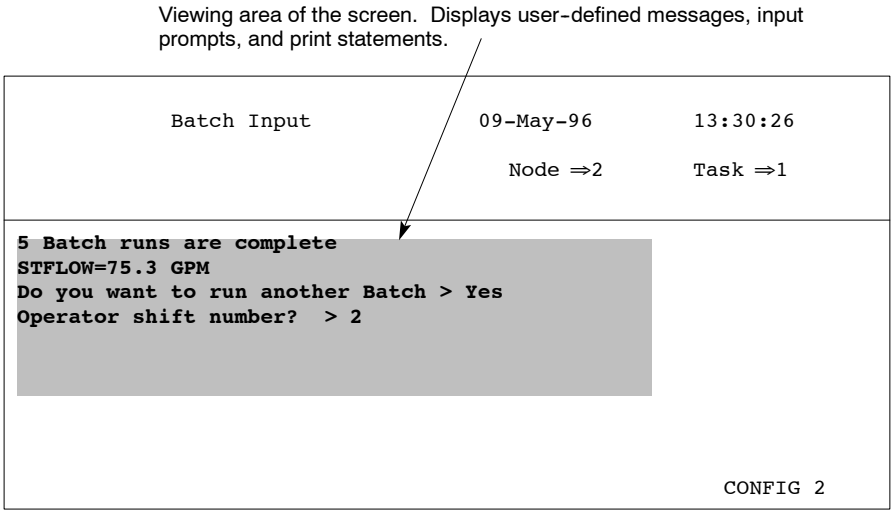
Screen data is generated by the following instructions: *prompt*, *reprompt*, *input*, *input$*, *print*, and *print$*. (For more information on these instructions, see Section 3: RBL Language.)

❑ **To call up the Batch Input screen:**

- If an alarm for the Input screen appears, press [ACTIVE ALARM]
  or
- If the task is not waiting for input, type:
  BAI (*node : task*) [ENTER]

The following message appears on the Batch Input screen:

No batch nodes seeking input



Batch Input Screen

Batch Log Screen

The Batch Log screen displays a log of entries from the execution of batch scripts.

❑ **To call up the Batch Log screen, type:**

BAL (*node* , *filename*) [ENTER]

❑ **To change the maximum number of log entries:**

- Cursor to the field, type in the number of log entries, and press [ENTER]. The default number of log entries is 1000; the maximum number of log entries is 2000.

❑ **To make a manual entry into the Batch Log file:**

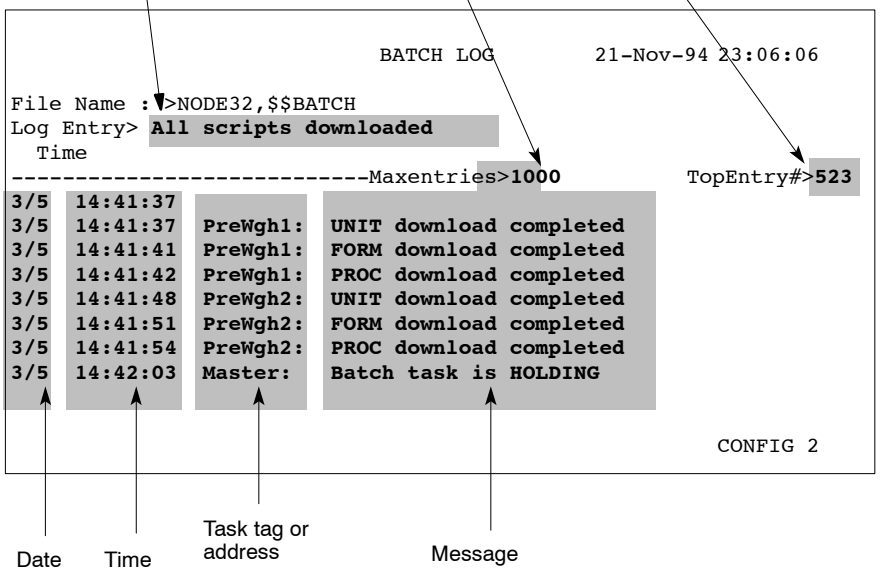
- Type a line of text and press [ENTER].

❑ **To display a log entry:**

- For a specific entry, cursor to the field, type in the number of the entry, and press [ENTER].

or

- For the latest entry, cursor to the field and press [ENTER].



Batch Log Screen

Section 5:
Scripts

Script Types

Batch recipes use library and start scripts:

Library script Defines either batch units or operations. For batch units, the library script defines aliases for equipment tags and addresses. For operations, the library script defines specific activities that are performed by the operation on the batch unit.

Start script Defines global procedures for all scripts that are used by the recipe. The start script executes in parallel with other recipe scripts. Values that you define in the start script remain in memory the entire time the recipe is running.

Batch tasks that are configured with the Batch Run screen use unit, formula, and procedure scripts:

Unit script Defines aliases for tags and addresses of the hardware devices that are used to make a batch of product.

Formula script Describes process endpoints, setpoints, and targets that are associated with making a batch of product.

Procedure script Defines the steps that used to make a batch product.

Calling Up Scripts (RBL File Contents Screen)

The RBL File Contents screen displays a list of scripts within an RBL file.

❑ **To call up the RBL File Contents screen, type:**

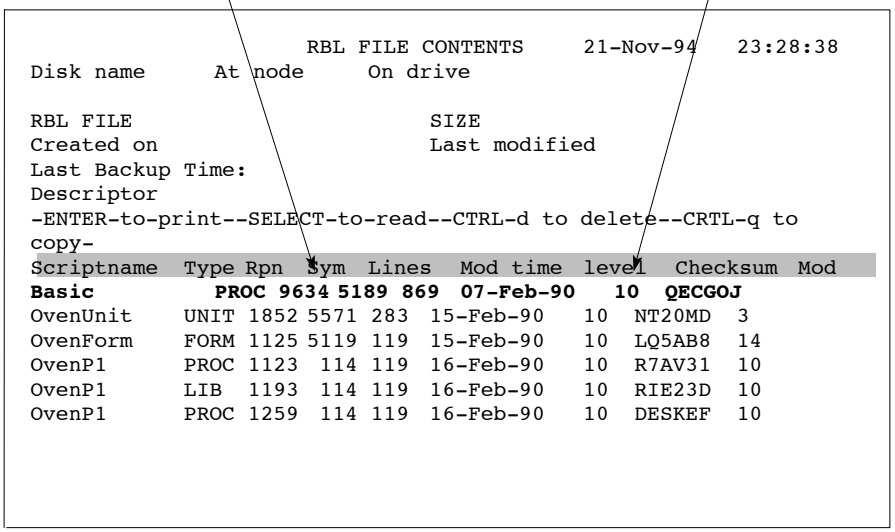
BAF (*node* , *filename*) [ENTER]

❑ **To call up a script:**

- Cursor to the script and press [SELECT]. Scripts are displayed on the Batch Script screen.

❑ **To print a script:**

- Cursor to the script and press [ENTER]. The script file is sent to the printer.



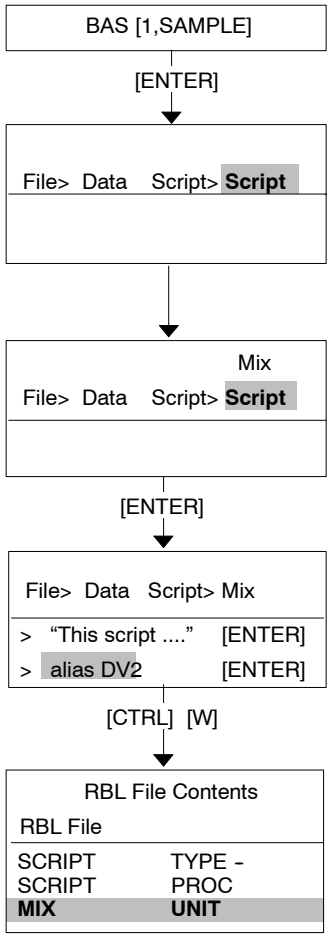
RBL File Contents Screen

Creating an RBL File and Initial Scripts

This procedure creates an initial RBL file and new scripts.

❑ **To create batch scripts:**

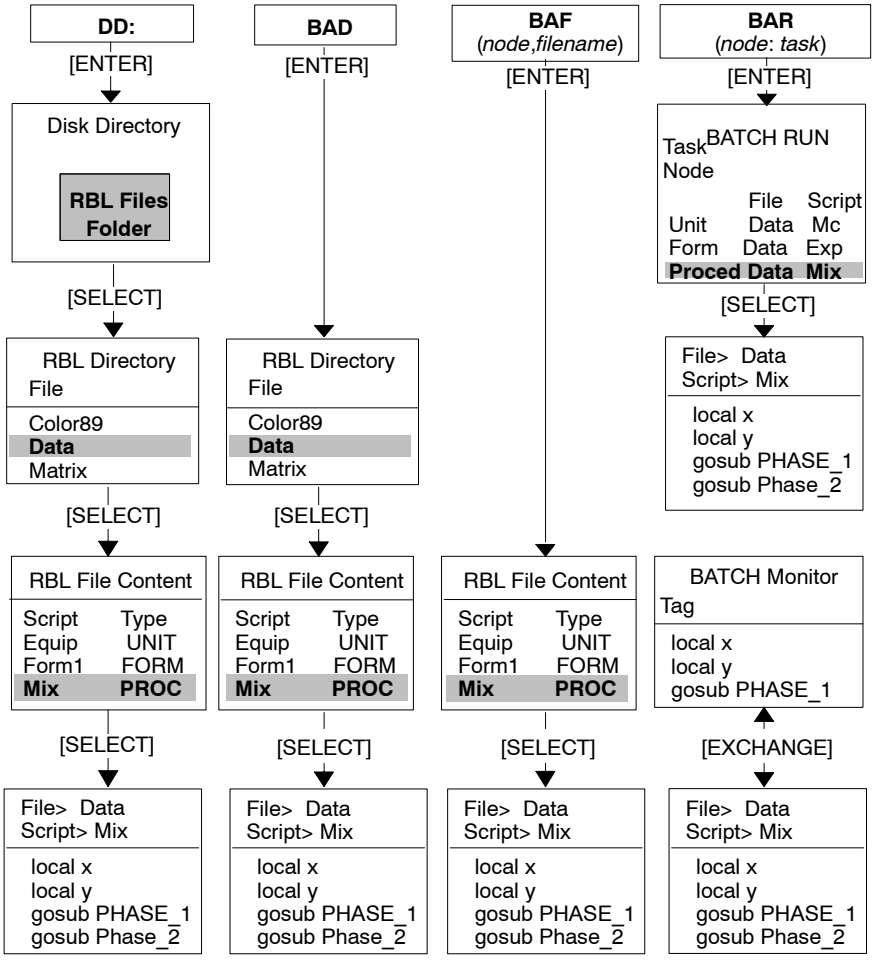
1. In the status line, type batch script command, volume, and script file name. Press [ENTER].
2. Cursor to the “Script” field.
3. Type a new script name. Press [ENTER].
4. Cursor below the line. Enter at least two lines of script code to put character space in the script. Blank lines also put space in the script.
5. Press [CTRL] [W] to save the script. The RBL File Contents screen lists two scripts: “SCRIPT“ and your new script.



Legend:		
		Move cursor to the highlighted field.
	[CTRL] [x]	Press CTRL key and the indicated key.
	[ENTER]	Press to call up display or enter value in field.

Calling Up Scripts

The following illustration shows several ways to call up scripts.



Legend:

[B] [x] [x] [x]	Type display mnemonic in status line. Move cursor to highlighted field.
[ENTER]	Press after typing command line mnemonic.
[SELECT]	Press to call up display.
[EXCHANGE]	Press to exchange screens

Editing Scripts

Use the following keys to edit a script. Hold down the [CTRL] key and press the indicated key at the same time. Search and scroll functions are entered from the display line in the upper left corner of the Batch Script screen.

Editing Functions

Activity	Steps
Enter text on a program line.	Cursor to the location and type the text characters. The program lines containing the characters turn yellow (default color). In insert mode, new characters are inserted to the left of the cursor. The line is automatically parsed when you press [ENTER]. In replace mode, new characters type over existing characters.
Toggle between text insert and replacement modes.	**[CTRL] T**
Insert an empty line above the existing line.	**[CTRL] I**
Delete a character.	**[CTRL] X**
Delete a line.	**[CTRL] D** The deleted line is placed in a delete buffer. You can delete a sequence of lines by pressing **[CTRL] D** repeatedly. Moving the trackball signals the end of the delete sequence. **CAUTION:** If you delete another line after moving the trackball, you will begin another delete sequence and clear the buffer of any lines that you deleted during the previous delete sequence.
Insert the delete buffer into the text.	**[CTRL] Y** All lines in the delete buffer are entered onto the line at the cursor.
Parse all unparsed lines in the script.	**[CTRL] P**

(continued on next page)

Editing Functions

Activity	Steps
Parse all unparsed lines and write the script to disk.	**[CTRL] W**
Undo changes made to the line last edited (before it is parsed).	**[CTRL] U**
Read a script from disk.	**[CTRL] R** Enter the volume and filename in “File” field, and the scriptname in “Script” field, and press [CTRL] [R].
Save the script as an “EDIT” type script. An “EDIT” file is useful for moving a script between a Batch script file and an RBLC.	**[CTRL] E**
Search forward in the script to find a character string.	Enter a slash mark (/) and the character string at the display line. For example: /VALVE1 [ENTER]
Search backward in the script to find a character string.	Enter a question mark (?) and the character string at the display line. For example: ?VALVE1 [ENTER]
Search and replace a character string.	Enter two character strings that are separated by slash marks (/) at the display line. For example: /VALVE 1/VALVE 2/ [ENTER] To confirm each search and replace, use the confirm parameter “c”. For example: /VALVE 1/VALVE 2/c [ENTER] To do a global search and replace, use the global parameter “g”. For example: /VALVE 1/VALVE 2/g [ENTER]
Scroll to the phase or label name.	Type the phase or label name at the display line and press [ENTER].
Scroll to the line number.	Type the line number at the display line and press [ENTER].

Section 6:
Rosemount Basic Language

Instructions

This section lists all the RBL instructions in alphabetical order with a short explanation of the purpose and use of each one. In the instruction syntax, instruction parameters are represented by alphabetic letters. Each parameter is explained in the “where” section. For more details about these instructions, see the *Rosemount Basic Language Manual*.

NOTE: Some instructions return status values that can be checked in the script to see if the instruction was successful. Status variables are designated as *rs* in the syntax.

RBL Instructions

abort

Triggers the execution of an *on abort* trap. If there is no *on abort* trap, the *abort* instruction is ignored.

abort_task(x)

Aborts a task and directs execution to an active *on abort* instruction. The *abort_task* instruction will cause a fatal error if there is no *on abort* trap in the script that is running.

where x is the tag name of the Batch Run screen or PeerWay address of a batch task (for example, 7702 or 77:02 for ControlFile 77, task 02).

rs=**acqfirst**(x,y)
rs=**acquire**(x,y)

Used by a master task to acquire a slave task. The *acquire* request is received by the slave task and placed in a queue of pending requests. The *acqfirst* instruction gives the master task request priority over all other *acquire* requests pending in the queue of the slave task.

where rs is a status variable. Values include:

- 1 Master task acquired slave task
- 0 Master task owns slave task
- -1 Request pending in slave task queue
- -2 Request was previously issued
- -3 Queue wait time has expired
- -4 Slave task is unavailable—value is returned only if an asterisk (*=no queue wait time) is specified in parameter y.

x is the batch tag of the task to be acquired.

y is an integer value for the queue time-out (for example, 5H). Values include:

- * = no queue wait time (asterisk)
- 0 = no queue time-out status returned
- S = seconds
- M = minutes
- H = hours
- D = days

(continued on next page)

RBL Instructions (continued)

rs=**acqwait**

Suspends execution of the slave task until a master task requests control with an *acquire* instruction. The slave task will not suspend execution if there is a pending request in its task queue.

where rs is a status variable. Values include:

1 Instruction was successful.

0 Instruction was unsuccessful.

alarm(x,y,z)

Generates an alarm of a specified alarm type and priority. You must define a string expression to appear in the alarm message.

where x is a numeric value (from 1 to 8) designating the alarm type to be generated.

1 Advisory alarm

2 Critical alarm

3 Hardware alarm

4 System alarm

5 Disk Event List alarm

6 Undefined alarm (causes runtime error)

7 Undefined alarm (causes runtime error)

8 Batch alarm

y is a numeric value designating the alarm priority. Priority values are from 1 (high) through 15 (low).

z is the string expression (maximum of 21 characters) that appears in the alarm message.

alias x:y

A user-defined name for a ControlBlock or I/O block address or subaddress.

where x is the name of the alias (maximum of 18 alphanumeric characters).

y is the tag name or address of the ControlBlock. The tag name is required when the alias is first defined, but not in subsequent declarations.

(continued on next page)

RBL Instructions (continued)

rs=**align_on**(“a”))<0
and
align_wait(x)

Synchronize the execution of the Main Recipe and Unit Recipe or the execution of two Unit Recipes. Two *align_on* instructions, one in each recipe, form a control pair. Neither recipe can advance beyond the *align_on* instruction until both recipes execute the *align_on* instructions.

The *align_on* instruction is used as an expression in a *while*, *until*, or *for* loop, as shown below. The *align_wait* instruction can suspend execution of the loop for a specified period. In this way, the function of the *align_wait* instruction is similar to a *sleep* instruction, except that the *align_on* instruction can interrupt the *align_wait* instruction.

```
while   ((rs=align_on(“hold”))<0
  align_wait(15)
```

NOTE:

- The *align_on* syntax must have double parentheses before rs and after “a”.
- You can use up to five *align_on* instructions per script. Each *align_on* instruction in the script must have a unique character string.
- You cannot nest an *align_on* loop in another loop.
- The *align_on* instruction cannot execute more than once every 10 seconds.

where rs is a status variable. Values include:

1 Instruction was successful.

-1 Instruction was unsuccessful.

a is a character string or string variable that matches *align_on* instructions in the two recipes. The *align_on* instructions for both recipes must have identical character strings. Neither recipe can advance until the strings in both *algin_on* instructions match.

x is a user-defined time interval in seconds.

(continued on next page)

RBL Instructions (continued)

almsg(x,y,z)

Generates a clear alarm for a specified alarm type and priority. You must define a string expression to appear in the alarm message.

where x is a numeric value (from 1 to 8) designating the alarm type to be generated.

1 Advisory alarm

2 Critical alarm

3 Hardware alarm

4 System alarm

5 Disk Event List alarm

6 Undefined alarm (causes runtime error)

7 Undefined alarm (causes runtime error)

8 Batch alarm

y is a numeric value designating the alarm priority. Priority values are from 1 (high) through 15 (low).

z is the string expression (maximum of 21 characters) that appears in the alarm message.

x$=**alv2str$(**y)

Returns a text string value for the access level of the user at the console.

where x$ is a string variable that receives a text string value from *alv2str*.

y is a number for the user access level that is entered in the Batch Log.

1 Any

2 Operator

3 Supervisor

4 Configuror

5 Recipe Manager

6 System Manager

(continued on next page)

RBL Instructions (continued)

a$=**andbit$**(b$,c$)

Performs an “and” operation on the binary bits of two string expressions. Returns a character byte value. If there is more than one character in a string expression, specify the character byte on which to perform the “and” operation (for example, (b$(2,2)). Otherwise, the “and” operation is performed on only the first byte in each string expression.

where a$ is a string variable that receives a character byte that is returned by *andbit$*.

b$ is a character string expression.

c$ is a character string expression.

array syntax
dim w(x,y)
stringdim w(x,y)
aliasdim v(x,y)

A user-defined name for an array of data values. In a one-dimensional array, each storage location is identified by a single index; in a two dimensional array, each storage location is identified by a row index and a column index.

where v is the name of the alias array.

w is the array name (maximum of 18 alphanumeric characters).

x is the number of storage locations in a one-dimensional array or rows in a two dimensional array. For example, x= 40 creates 40 storage locations or rows indexed 0 to 39.

y is the number of columns in a two-dimensional array. For example, y=40 creates 40 columns indexed 0 to 39.

(continued on next page)

RBL Instructions (continued)

rs=**array_copy**(r,s,t,u,v,w,x,y,z)

Copies array elements into another array. **NOTE:** You cannot use *array_copy* to copy string arrays and redundant virtual arrays (*stringdim*, *vstringdim*, *rdim*, and *rstringdim*).

where rs is a status variable.
- 1 Instruction was successful.
- 0 Instruction was unsuccessful.

r is the source array from which data is copied.

s is the destination array to which data is copied.

t is the array offset in columns in the destination array.

u is the beginning row in the source array from which data is copied.

v is the beginning column in the source array from which data is copied.

w is the ending row in the source array from which data is copied.

x is the ending column in the source array from which data is copied.

y is the beginning row in the destination array to which data is copied.

z is the ending column in the destination array to which data is copied.

rs=**array_init**(r,s,t,u,v,w,x,y,z)

Initializes array elements to a specified value.

where rs is a status variable.
- 1 Instruction was successful.
- 0 Instruction was unsuccessful.

r is the source array which is initialized.

s is the initialization value.

t is the array offset in columns in the initialized array.

u is the row at which to begin initialization.

v is the column at which to begin initialization.

w is the row at which to end initialization.

x is the column at which to end initialization.

(continued on next page)

RBL Instructions (continued)

a=**asc**(b$)

Converts the first character of a string to the decimal ASCII representation of the character (0–255).

where a is a variable that receives the ASCII value.

b$ is a string expression.

backup

Causes program execution to hold until it is backed up to the nonvolatile memory.

x=**badunit**

Returns the plant unit number for a ControlBlock if:

- A Working Recipe attempts to write to a ControlBlock with a plant unit other than zero (from 1 to 255).
- The Batch Configuration screen has been configured to enforce plant units.
- The Working Recipe does not own the batch plant unit.

where x is a variable that receives the plant unit number from the *badunit* instruction.

(continued on next page)

RBL Instructions (continued)

rs=**begin_recipe**("v","w","x",y,z_1...z_{10})

Starts the execution of a configured Control Recipe. The Control Recipe must have been validated and assigned a batch unit(s).

where rs is a status variable. Values include:

-14 Invalid number for batch unit set. The number must be between 1 and 30.

-13 Write to recipe locals failed.

-12 Batch ID is out of range. Check the syntax.

-11 Recipe software version is incompatible with CP software.

-10 Insufficient volatile memory to run the recipe.

-9 Control Recipe has not been validated.

-8 Fatal or internal software error encountered.

-7 Recipe is missing a parameter.

-6 Request message failed due to internal software error.

-5 Control Recipe cannot access disk volume.

-4 Control Recipe cannot access batch unit.

-3 Task cannot access the recipe on the disk.

-2 Recipe name or Batch ID is bad. Another recipe might be using the Batch ID.

-1 Another batch is using the task that is assigned to the batch unit.

1 Recipe started successfully.

2 Recipe started successfully from a backup volume. Primary volume is missing.

(continued on next page)

RBL Instructions (continued)

v is the name of the Control Recipe. Enter either as a character string or string variable.

w is the Batch ID for the Control Recipe. Enter a character string or string variable up to 32 characters. If the Batch ID is generated by *wait_bid* and *send_bid* instructions, enter an empty character string " ".

x is the name of a formula (optional). Enter either as a character string or string variable. If you do not want to include a formula, enter an empty character string " ".

NOTE: You must assign a Formulas Table to the Control Recipe before you can use formulas.

y is the index number of a batch unit set from 1 to 30 (optional). If you do not want to include a batch unit set, enter 0.

NOTE: You must assign a Unit Set File and unit set elements to the Control Recipe before you can use formulas.

z Is the number of a recipe local value (optional). You can include up to 10 recipe locals. If you do not want to include a recipe local, enter 0.

begin_task(x) Starts a task that is in the idle or finished state.

where x is the tag name of the Batch Run screen or PeerWay address of a batch task (for example, 7702 or 77:02 for ControlFile 77, task 02).

break Transfers control from an indented program line in a *for*, *while*, or *until* instruction out to the next level of indentation.

bumpid Increments the last character of the batch ID string. If the last character rolls over, the next to the last character is incremented.

(continued on next page)

RBL Instructions (continued)

rs=**byte2flag**(a$(u,v),w,x,y)

Writes two characters of a string variable to the 16 user flags (discrete outputs a to p) of a ControlBlock.

where rs is a status variable. Values include:

1 Character string is empty of values.

2 Illegal block number used.

4 Function successfully executed.

a$ is the character string expression.

u is the index of the first character in the character string expression. Flags i through p represent the bit configuration (in binary notation 0 or 1) of the first character.

v is the index of the second character in the character string expression. Flags a through h represent the bit configuration (in binary notation 0 or 1) of the second character.

w is the base alias or address for the ControlBlock from which the user flag values are written to by the function.

x is an integer or variable value that offsets the value of the base ControlBlock address (parameter w). The offset can be less than or greater than the base block address.

y is an integer or variable value that corresponds to the ControlBlock analog register being written to. The integers 0 through 15 correspond to register Q and A through O.

chain(x,y)

Suspends execution of a current or parent script and starts execution of a linked or child script.

where x is the name of the RBL file or a string variable that contains the name.

y is the name of the script that is being chained to or a string variable that contains the name.

a$=**chr$**(b)

Converts a decimal number to its equivalent ASCII character. This function is the inverse of the asc function.

where a$ is a character variable that receives the character value.

b is an ASCII numeric value (0–255).

(continued on next page)

RBL Instructions (continued)

clear(x,y,z)

Clears a specified alarm type and event function. You must define the string expression to appear in the alarm message.

where x is a numeric value (from 1 to 8) that designates the alarm type that is cleared.

1 Advisory alarm

2 Critical alarm

3 Hardware alarm

4 System alarm

5 Disk Event List alarm

6 Undefined alarm (causes runtime error)

7 Undefined alarm (causes runtime error)

8 Batch alarm

y is a numeric value (1-15) designating the alarm priority. Priority values are from 1 (high) through 15 (low).

z is the string expression (maximum of 21 characters) that appears in the alarm message.

rs=**close**(x)

Closes an active report file.

where rs is a status variable. Values include:

1 Instruction was successful.

0 Instruction was unsuccessful.

x is a report reference number (created by an *open* instruction) from 1 to 10.

constant a:x

A user-defined name for a floating point number that does not change during execution of the task.

where a is the name of the constant (maximum of 18 characters).

x is a floating point number (required when constant is first defined, but not in subsequent declarations).

continue

Transfers control from the *continue* instruction line to the *for* or *while* program line.

(continued on next page)

RBL Instructions (continued)

cont_task(x)

Continues a task that is suspended by a *halt* instruction.

where x is the tag name of the Batch Run screen or PeerWay address of a batch task (for example, 7702 or 77:02 for ControlFile 77, task 02).

rs=**dequeue**(x)

Removes a pending *acquire* request from the slave task queue when executed by the master task. Before using it, thoroughly read the material about task synchronization in the Batch Software and RBL Controller Manual, Chapter 2: RBL Language, Section 9.

where rs is a status variable. Values include:

1 Instruction was successful.

0 Instruction was unsuccessful.

x is the tag of the slave task for which an acquire request has been issued.

dishold

Disables an operator "hold" command. The "hold" command is ignored while it is disabled. However, a "hold" command that is entered while it is disabled will execute later when it is enabled.

disownunit(x)

Terminates the Working Recipe ownership of a specified batch plant unit. You can check the ownership status on the Batch Plant Unit Status screen.

where rs is a status variable. Values include:

x is a number of a plant unit (between 1 to 255) or a variable for the plant unit number.

display("x",y)

Designates variables, constants, aliases, and dim arrays for display on the Working Recipe, Batch Monitor screen, Batch Recipe Viewing screen and Process Graphic.

where x is a line of text that can be displayed on the batch screen.

y is a variable that can be displayed on the batch screen.

end

Terminates execution of the script.

endchain

Terminates a chain operation in which two scripts are linked by a *chain* instruction. The task resumes execution on the line after the *chain* instruction in the original parent script.

enhold

Enables an operator "hold" command. A "hold" command that is entered while it is disabled will execute when it is enabled.

(continued on next page)

RBL Instructions (continued)

event(x,y,z)

Creates an entry in an event list for a specified event type and function. You must define a character string expression to appear in the event list.

where x is a numeric value (0-255) designating the event type.

y is a numeric value (0-15) designating the event priority. This parameter is not useful at this time. Enter any number.

z is the string variable or the entire string (maximum of 21 characters) that is to appear in the event list. Enclose the string within quotation marks.

exit(x)

Terminates the execution of a script. It is also used to select one of 16 possible alternative paths that are introduced by a Decision icon in the recipe. The Decision icon chooses the path that is specified by the *exit* instruction.

where x is a number from 1 to 16 of a path that is configured for the Decision icon. A variable or alias can represent the path number.

(continued on next page)

RBL Instructions (continued)

x=**fetch**(“y”,“z”,rs)
x$=**fetch$**(“y”,“z$”,rs)

Read variables in another script. The fetch instruction allows the recipe to transfer values between two Unit Recipes or a Unit Recipe and Main Recipe. Comm-Op icons are used to execute the *fetch* or *fetch$* instruction. Each Comm-Op icon in the Comm-Op icon pair is linked by an operation to a script. The *fetch* or *fetch$* instruction in one of the scripts reads a variable from the other script.

NOTE:

- Execution time for the *fetch* or *fetch$* instruction might vary. If you want to use these instructions to read variables at a specific point in the process, use the *align_on* instruction to suspend execution of the script read.
- In order to make sure that both subrecipes are running, use the *fetch* or *fetch$* instruction after an *align on* instruction.
- To prevent excessive PeerWay messages and use of processor memory, the *fetch* or *fetch$* instruction should not execute more than once every 10 seconds.

where rs is a status variable. Values include:

-7 Failed: Unknown error.

-6 Failed: Incompatible hardware and software.

-5 Failed: Variable not found.

-4 Failed: Other task not found.

-3 Failed: Other Unit Recipe does not have the same Batch ID.

-2 Failed: Unit Recipe not found.

-1 Failed: Comm channel not found.

1 Variable returned successfully.

x is a variable that receives the value of variable z. Use a local variable for *fetch*. Use a string variable for *fetch$*.

y is the symbol type of variable read, either “shared” or “private.”

z is the name of the variable read. *fetch* reads a local variable. *fetch$* read a string variable.

(continued on next page)

RBL Instructions (continued)

rs=**file_copy**(v,“w”,“x”,“y,“z”)

Copies a virtual array file (*vdim*), virtual array of strings (*vstringdim*), or a report file from a source file to a destination file of the same file type. *file_copy* cannot copy redundant virtual arrays (*rvdim* and *rvstringdim*).

where rs is a status variable.

v is the folder type in which the file is located. Values Include:

1 SRU DATA

2 REPORTS

w is the source volume in which the source file is located. The volume can be either a character string or a string variable.

x is the name of the source file. The source file can be either a character string or a string variable.

y is the destination volume where the destination file is copied. The volume can be either a character string or a string variable.

z is the destination file. The destination file can be either a character string or a string variable.

(continued on next page)

RBL Instructions (continued)

rs=**file_delete**(v,“w”,“x”)

Deletes a virtual array file (*vdim*), virtual array of strings (*vstringdim*), or a report file. *file_copy* cannot copy redundant virtual arrays (*rvdim* and *rvstringdim*).

where rs is a status variable.

v is the folder type in which the file is located. Values Include:

1 SRU DATA

2 REPORTS

w is the source volume in which the file you want to delete is located. The volume can be either a character string or a string variable.

x is the name of the file you want to delete. The file can be either a character string or a string variable.

rs=**file_rename**(v,“w”,“x”,“y”)

Renames a virtual array file (*vdim*), virtual array of strings (*vstringdim*), or a report file. *file_copy* cannot copy redundant virtual arrays (*rvdim* and *rvstringdim*).

where rs is a status variable.

v is the folder type in which the file is located. Values Include:

1 SRU DATA

2 REPORTS

w is the source volume in which the file you want to rename is located. The volume can be either a character string or a string variable.

x is the name of the file you want to rename. The file can be either a character string or a string variable.

y is the new name of the file. The file can be either a character string or a string variable.

filerr

Is used as an expression in a *while* instruction to test the execution status of an *on file_n_err* trap. For more information on *filerr*, see RB 1-7.

(continued on next page)

RBL Instructions (continued)

finish — Terminates the script at the finish script line.

a$(u,v)=**flag2byte$**(w,x,y,r,s) — Reads the 16 user flags (discrete outputs a to p) from a ControlBlock and writes them as two characters of a string variable.

where
- a$ is the character string expression.
- u is the index of the first character in the string expression. Flags i through p represent the bit configuration (in binary notation 0 or 1) of the first character.
- v is the index of the second character in the string expression. Flags a through h represent the bit configuration (in binary notation 0 or 1) of the second character.
- w is the alias or address for the ControlBlock from which the user flag values are read by the function.
- x is an integer or variable value that offsets the value of the ControlBlock address (parameter w). The offset can be less than or greater than the current block address.
- y is an integer or variable value that corresponds to the ControlBlock analog register being read from. The integers 0 through 15 correspond to register Q and A through O.
- rs is a status variable. Values include:
  - 1 Character string is empty of values.
  - 2 Illegal block number used.
  - 4 Function successfully executed.

(continued on next page)

RBL Instructions (continued)

flog(“x”,y,a$)

Allows print messages and optional format requirements to be made from a batch script to a batch log file identified by a unique number.

where x is the batch log file name or a string variable that contains the name.

y is the print message that is to be copied to the log file and the optional format requirements for that message.

a$ is an optional variable whose value is converted to a string value and included in the print message at the % designator (see Formatted Print Options).

for start,stop,count

<instructions>

Executes indented lines a specified number of times:

- start – is the initial expression for the counter.
- stop – is the test expression for the counter.
- count – is the control expression for counting the number of times the *for* loop is executed.

Example: for j=1,j<11,j=j+1

<instructions>

a=**getbit**(b$,c)

Returns the binary state (0 or 1) of a specified bit in a character byte.

where a is a variable that receives the state value (0 or 1) of the bit.

b$ is the string expression containing the byte. If a string is specified, the first byte of the string is used.

c is the numeric bit position (0–7).

(continued on next page)

RBL Instructions (continued)

rs=**getmaterial**(x$,y$,z) Reads property values of a material in the Batch Materials screen.

where rs is a status variable. Values include:

0 Instruction was successful.

-1 Value not found; system or disk problem.

-2 Material name not found.

-3 Property name not found.

x$ is the name of the material that is listed in the Batch Materials Table.

y$ is the name of the material property.

z is a variable that receives the property value.

rs=**getmatlim**(v$,w$,x,y,z) Reads the property value limits of a material in the Batch Materials screen.

where rs is a status variable. Values include:

0 Instruction was successful.

-1 Value not found; system or disk problem.

-2 Material name not found.

-3 Property name not found.

v$ is the name of the material in the Batch Materials Table.

w$ is the name of the material property.

x is a variable that receives the property value.

y is a variable that receives the low limit for the property value.

z is a variable used to get a high limit for a property value.

a=**getssm**(x) Converts a realtime value into seconds since midnight.

where a is a variable or array element that receives a converted realtime value in seconds since midnight.

x is a variable or array storage location that contains a realtime value.

(continued on next page)

RBL Instructions (continued)

a=**getsss**(x)

Converts a realtime value into seconds since Sunday (sss) starting at midnight.

where a is a variable or array element that receives a converted realtime value in seconds since Sunday.

x is a variable or array storage location that contains a realtime value.

gettime(rt,n1,n2,n3,n4, n5,n6)

Converts a realtime value into arithmetic values for year, month, day, hour, minutes, and seconds.

where rt is the variable or array storage location that contains a realtime value.

n are variable or array storage locations that receive a converted realtime value. Date and time values are assigned in the following order: year, month, day, hour, minute, second. Variables or array indices are required for all six values (n1,n2,n3,n4,n5,n6).

a=**getttrend**(b,c,d,e,f)

Returns data points from an SRU trend file and writes these data points in a data array.

a is a variable that indicates the number of data points that are returned by *gettrend*.

b is the ControlBlock link for a trend file from which data points are returned.

c is the PeerWay node number on which the trend file resides.

d is the number of data points that are displayed in the trend group to the trend file.

e is the two-dimensional tracking array in which time stamps are stored. Specify the row and column numbers for the tracking array. Column dimensions are optional.

f is the data array in which trending data points are stored. Specify the row and column numbers for the data array. Column dimensions are optional.

(continued on next page)

RBL Instructions (continued)

global variable syntax
global x

A user-defined name for a global variable. A global variable must be declared as a local variable in the first batch task in the CP or as a global variable in other batch tasks that use that variable.

where x is the name of the local variable that is declared in an associated task. The associated task must be running before the global variable can be assigned a value.

gosub x

Directs execution of a script to a phase or label name. Labels must be in the same phase as the *gosub* instruction or in the global environment of the script.

where x is the phase name (for example, ACT1) or label name (for example, _SUBACT)

goto x

Directs execution of the script to a label name.

where x is the label name (for example, _SUBACT).

halt

Suspends execution of the script until you execute a “Cont” command on the Batch Monitor screen or BFACE object.

halt_task(x)

Halts the execution of a task until a *cont_task* instruction or “Cont” command is executed.

where x is the tag name of the Batch Run screen or PeerWay address of a batch task (for example, 7702 or 77:02 for ControlFile 77, task 02).

hold

If enabled, triggers an *on hold* trap. The *enhold* instruction enables the *hold* instruction. The *dishold* instruction disables the *hold* instruction. If there is no *on hold* trap, the *hold* instruction is ignored.

holding

Returns a “yes” or “no” value to indicate if a holding flag has been set by the *hold* instruction.

(continued on next page)

RBL Instructions (continued)

if expression
<instruction>

If expression is:

- True - Execute the instruction.
- False - Do not execute indented instructions.

Example:
```
if STPT<5
    <instruction>
```

if expression
<instruction1>
else
<instruction2 >

If expression is:

- True - Execute instruction1 and skip instruction2. Instruction2 is not executed.
- False - Skip instruction1 and execute instruction2.

Example:
```
if FLOW>120
    <instruction1>
else
    <instruction2>
```

if expression1
<instruction1>
elseif expression2
<instruction2>
elseif expression3
<instruction3>

If expression1 is:

- True - Execute instruction1.
- False - Evaluate expression2.

Else if expression2 is:

- True - Execute instruction2.
- False - Evaluate expression3.

Else if expression3 is:

- True - Execute instruction3
- False - Skip instruction3.

Example:
```
if  TEMP<=78
<instruction1>
elseif (TEMP>78)&(TEMP<=100)
<instruction2>
elseif (TEMP>100)&(TEMP<=144)
<instruction3>
```

(continued on next page)

RBL Instructions (continued)

input(“x”,y)

Prompts a user for numeric input from the Batch Input screen. Copies an optional text message to the screen and writes operator input to a variable. It must be indented under a *prompt* instruction.

where x is the message that is to be copied to the Batch Input screen and the optional format requirements for that message.

y is a numeric variable that defines the operator-enterable field on the Batch Input screen. Also accepts “Y” (yes) and “N” (no) as inputs.

x$=**input$**(“y”)

Prompts a user for character input from the Batch Input screen. Copies an optional text message to the screen and writes operator input to a string variable. Must be indented under a *prompt* instruction.

where x$ is a string variable that receives data from the operator-enterable field on the Batch Input screen.

y is the message that is to be copied to the Batch Input screen and the optional format requirements for that message.

(continued on next page)

RBL Instructions (continued)

rs=**inputp**(“x”,y,z)

Assigns permission to use subsequent code in the script. If *inputp* fails, no script after *inputp* will execute.

where rs is a status variable. Values include:

- 1 Instruction was successful.
- -1 Console does not have a password configured.
- -2 User does not exist.
- -3 Problem with the password file.
- -4 Incorrect password.
- -5 Incorrect user access level.

x is a login name. It can be either a character string or a string variable of up to 16 characters.

y is the user permission level that is required. The user permission is entered into the Batch Log.

- 1 Any
- 2 Operator
- 3 Supervisor
- 4 Configuror
- 5 Recipe Manager
- 6 System Manager

z Option is not available. Enter 0.

kill_task(x)

Kills the execution of a task and puts it in the IDLE state.

where x is the tag name of the Batch Run screen or PeerWay address of a batch task (for example, 7702 or 77:02 for ControlFile 77, task 02).

(continued on next page)

RBL Instructions (continued)

label syntax

A user-defined label name of a program line. Labels are useful for identifying portions of the script.

_x:

where x is the name of the label. All label names must be preceded by an underscore character (for example, _MIX1).

local variable syntax
local x

A user-defined name for a data value that may change in the batch task. The local variable can be used in the same task to pass values between scripts.

where x is the name of the variable.

rs=**mapunit**(x)

Assigns a batch unit set to the Working Recipe. You must assign a Unit Set File and unit set elements to the Control recipe before using *mapunit*.

where rs is a status variable. Values include:

- 0 Instruction was successful.
- -1 Invalid number. Batch unit set must be between 1 and 30.
- -2 Batch unit set failed validation.
- -3 Internal error while executing mapunit.
- -4 Volume search failed.

x is the number (from 1 to 30) of the batch unit set.

rs=**masterof**(x)

Returns a status value to the master task to determine if the task is master of a specified slave task.

where rs is a status variable. Values include:

- 1 This task is the master.
- 0 This task is not the master.

x is the tag of the slave task being checked.

(continued on next page)

RBL Instructions (continued)

rs=**materialdesc**(x$,y$)

Reads the material description of a specified material in the Batch Materials Table and writes the description to a character string variable.

where rs is a status variable. Values include:

- 0 Instruction was successful.
- -1 Value not found; system or disk problems.
- -2 Material name not found.
- -3 Property name not found.

x$ is a character string variable for the name of a material in the Batch Materials Table.

y$ is a character string variable that receives the text description of the material that is specified by x$.

rs=**materialunits**(x$,y$, z$)

Reads the engineering units of measure of a specified material property in the Batch Materials Table and writes the units of measure to a character string variable.

where rs is a status variable. Values include:

- 0 Instruction was successful.
- -1 Value not found; system or disk problems.
- -2 Material name not found.
- -3 Property name not found.

x$ is a character string variable for the name of a material in the Batch Materials Table.

y$ is a character string variable for the name of a material property in the Batch Material Properties screen.

z$ is a character string variable that receives the engineering units of measure of the material property specified by x$ and y$.

next

Terminates execution of a conditional *while*, *for*, or *until* instruction when an operator executes a “NEXT” command. The loop terminates at the *next* instruction line.

(continued on next page)

RBL Instructions (continued)

a=**nil**	Assigns a blank “nil” value to a data array storage location. The “nil” value cannot be used in arithmetic operations. where a is the variable or array storage location that is assigned the “nil” value.
a=**nvfree**	Assigns a value for the amount of available nonvolatile memory to a variable. where a is a variable that receives the nonvolatile memory value.
on x	Executes if a ControlBlock logic step or continuous output goes into alarm or causes an event alarm. where x is an alias, tag, or address to identify the ControlBlock alarm.
on abort	Executes in response to an *abort* instruction in the script or an “abort” command from the Batch Monitor screen, RBL Monitor screen, Batch Run screen, or BFACE object.
on acquire_err	Executes in response to a faulty acquire condition. A faulty condition can occur if the master task cannot log an acquire request to the acquire queue of the slave task.
on chain_err	Executes if the script that is named by a *chain* instruction does not exist on either the primary or backup volume when the *chain* instruction is executed.
on file_n_err	Executes if an attempt to print a report fails. where n is the number (1–10) of the open report.
on input_err	Executes in response to an incomplete execution of an *input* instruction during a prompt sequence.
on link_err	Executes if an error occurs when the script is accessing a ControlBlock.
on lost_master	Executes if the slave task loses its master task. The slave task can use *on lost_master* to release itself from a lost master task.
on lost_slave	Executes if the master task loses its slave task. The master task can use *on lost_slave* to unacquire a lost slave task or remove an entry for a lost slave task from its acquire queue.
on nest_err	Executes if outstanding nesting instructions exceed nesting limits (maximum of 40 active nesting instructions).
on no_backup	Executes if the batch task is not being backed up to the nonvolatile memory.
on power_up	Executes in response to a restart of the Coordinator Processor RBL Controller, or a load switch to a redundant CP card.

(continued on next page)

RBL Instructions (continued)

on stop — Executes in response to the execution of a *stop* instruction in the script, or a "stop" command from the Batch Monitor screen, the RBL Monitor screen, the Batch Run screen, or BFACE object. Do not use on stop in a unit or formula script.

on task_retry — Executes if a batch task cannot be accessed on the PeerWay by any of the acquire/release instructions.

on unacquire — Executes if the master task terminates the master/slave relationship. Use this trap only in the slave task. If there is no *on unacquire* trap in the slave task, a fatal error will result if the master task fails.

on unowned_unit — Executes if:

- A Working Recipe attempts to write to a ControlBlock with a plant unit other than zero (from 1 to 255).
- The Batch Configuration screen has been configured to enforce plant units.
- The Working Recipe does not own the batch plant unit.
- The plant unit is not specified for the batch unit on the Batch Units Table.

on varray_err — Executes if a file error prevents the script from either reading or writing to a virtual array. The *on varray_err* will not detect errors with virtual redundant or string arrays (*vstringdim*, *rvdim*, or *rvstringdim*).

rs=**open**(w,x,y,z) — Opens the report file for one of 10 active report file names on a specified disk volume name.

where
- rs — is a status variable. Values include:
  - 1 Instruction was successful.
  - 0 Instruction was unsuccessful.
- w — is a number from 1 to 10 specifying one of 10 active report file names.
- x — is the volume name of the disk (y may be defaulted or entered as node:drive).
- y — is the report file name.
- z — is an optional append action performed by the *open* instruction. Options include: 0 to open a new report (default), 1 to append a report file.

(continued on next page)

RBL Instructions (continued)

a$=**orbit$**(b$,c$)

Performs an "or" operation on the binary bits of two string variables. Returns a character byte value.

where a$ is a string variable that receives the character byte value.

b$ is a character string expression. If there is more than one character in the string expression, specify the character byte on which to perform the "or" operation (for example, (b$(2,2)).

c$ is a character string expression. If there is more than one character in the string expression, specify the character byte on which to perform the "or" operation (for example, (c$(128)).

opremain=x

Displays the expected execution time remaining in an operation.

where x is the time value that is assigned to *opremain*. *opremain* counts backwards from this time value to 0. You can use *opremain* with the *display* instruction to display time or with a *while* instruction to suspend a script.

rs=**ownunit**(x)

Assigns ownership of a specified batch plant unit to a Working Recipe. You can check the ownership status on the Batch Plant Unit Status screen.

where rs is a status variable. Values include:

1 Instruction was successful.

0 Instruction was unsuccessful.

x is a number of a plant unit (between 1 to 255) or a variable for the plant unit number.

phase syntax

A user-defined phase name of a program line that divides the script into sections.

x:

where x: is the phase name. All phase names must end in a colon (for example, LOAD:)

phremain=x

Displays the expected execution time remaining in a phase.

where x is the time value that is assigned to *phremain*. *opremain* counts backwards from this time value to 0. You can use *phremain* with the *display* instruction to display time or with a *while* instruction to suspend a script.

(continued on next page)

RBL Instructions (continued)

print(x,"y",z)

Copies a print message to an active report file or a Batch Input screen (*). The character string can contain an optional format specification. An optional variable can be included for conversion to a character string (in which case, you must use a format specification).

where x is a number (1–10) referring to the active report file or an asterisk (*) that designates that the print message is to be sent to the Batch Input screen. If using an asterisk, the *print* Instruction must be indented under a *prompt* instruction.

y Is the print message that is copied to the report file or Batch Input screen and the optional format parameters for that message.

z is an optional variable whose value is converted to a string value and included in the print message at the % designator (see Formatted Print Options).

x$=**print$**("y",z **)**

Writes a print message to a string variable. The print message can contain an optional format specification. An optional variable can be included for conversion to a character string (in which case, you must use a format specification).

where x$ is a string variable to which the *print$* instruction writes the print message as a character string.

y is the message that is written to a string variable and the optional format parameters for that message.

z is an optional variable whose value is converted to a string value and included in the message at the % designator (see Formatted Print Options).

priority(x)

Assigns a priority number to the script that determines how fast the script is executed or execution speed of the task in relation to other tasks running on the same node. For more information on priority, see RB: 1–6.

where x is a number from 0 to 1000. The larger the number, the faster the maximum possible execution speed. The *priority* number can be specified as an integer or represented by a variable, alias, or constant.

(continued on next page)

RBL Instructions (continued)

prompt

Initiates communication between the batch script and the Batch Input screen. It accesses the Batch Input screen by issuing an alarm. You must indent all lines that are used for communication under the *prompt* instruction.

rs=**putmaterial**(x$,y$,z)

Writes property values to a material in the Batch Materials screen.

where rs is a status variable. Values include:

- 0 Instruction was successful.
- -1 Value not found; system or disk problem.
- -2 Material name not found.
- -3 Property name not found.

x$ is the name of the material in the Batch Materials Table.

y$ is the name of the material property.

z is a variable that *putmaterial* writes to a property value.

jt=**puttime**(rt,n1,n2,n3, n4,n5,n6)

Converts time values for year, month, day, hour, minutes, and seconds into a realtime value.

where jt is a variable or array storage location to which gettime can assign the realtime value.

rt is the variable or array storage location that contains the realtime value.

n are variables or array storage locations that contain time values to be converted to a realtime value. Date and time values are assigned in the following order: year, month, day, hour, minute, second. Variables or array indices are required for all six values (n1,n2,n3,n4,n5,n6).

quietlog

Filters messages so that only messages that report run time errors are sent to the Batch Log.

a=**realtime**

Returns the current time of day in realtime notation. Realtime values must first be converted to decimal or floating point notation to be used in arithmetic operations.

where a is the variable or array storage location that receives the realtime value.

(continued on next page)

RBL Instructions (continued)

recabort Triggers the execution of *on abort* traps in all scripts that are associated with the recipe. If there are no *on abort* traps, *recabort* is ignored.

rechold Triggers execution of *on hold* traps in all scripts that are associated with the recipe. If there are on *on hold* traps, rechold is ignored.

recipe_local(x) Reads or writes to recipe local values in the Working Recipe. You can also use a *begin_recipe* instruction to assign recipe local values to a recipe.

where x is an index number from 1 to 10 for a recipe local value.

recresume Continues all scripts that are associated with the recipe and are suspended by *on hold* traps.

rs=**release** Releases the slave task from control of a master task when executed by the slave task.

where rs is a status variable. Values include:

1 Master task is suspended by a *relwait* instruction.

0 Master task is not suspended by a *relwait* instruction.

rs=**relwait**(x) Suspends the execution of the master task until the slave task issues a *release* instruction. It is executed by the master task.

where rs is a status variable. Values include:

1 Instruction was successful.

0 Instruction was unsuccessful or slave task was terminated while the master task was suspended.

x is the tag of the slave task.

reprompt Initiates communication between the batch script and the Batch Input screen. It performs the same function as the *prompt* instruction except that it does not issue a batch prompt alarm while the Batch Input screen is communicating with the task.

resume Continues a script that is suspended by an *on hold* trap.

(continued on next page)

RBL Instructions (continued)

resume_task("x")

Continues a script in a task that is suspended by an *on hold* trap.

where x is the Batch Run screen task or PeerWay address of a batch task (for example, 7702 or 77:02 for ControlFile 77, task 02).

retry x

Directs execution from within a conditional loop to a label in the script when an operator executes a "Retry" command.

where x is a label name in the script. Labels must begin with an underscore, for example _LABEL.

return

Used with a *gosub* instruction or *on* instruction to return execution to the program line after the *gosub* or *on* instruction.

(continued on next page)

RBL Instructions (continued)

rs=**run_recipe**(“x”,“y”,z) Starts the execution of a configured Control Recipe. The Control Recipe must have been validated and assigned a batch unit(s).

where rs is a status variable. Values include:

- -9 Control Recipe has not been validated
- -8 Fatal or internal software error encountered.
- -7 Recipe is missing a parameter.
- -6 Request message failed due to internal software error.
- -5 Control Recipe cannot access disk volume.
- -4 Control Recipe cannot access batch unit.
- -3 Task cannot access the recipe on the disk.
- -2 Recipe name or Batch ID is bad. Another recipe might be using Batch ID.
- -1 Another batch is using the task that is assigned to the batch unit.
- 1 Recipe started successfully.
- 2 Recipe started successfully from a backup volume. Primary volume missing.

x is the name of the Control Recipe. Enter either as a character string or string variable.

y is the Batch ID for the Control Recipe. Enter a character string or string variable up to 32 characters. If the Batch ID is generated by *wait_bid* and *send_bid* instructions, enter an empty character string “ ”.

z Indicates whether the recipe that is started by *run_recipe* should use the same batch log as the recipe that executes *run_recipe*. Enter 1 for yes or 0 for no.

(continued on next page)

RBL Instructions (continued)

x=**scdp**(y) Assigns the number of decimal places of a ControlBlock continuous input or output to a declared variable.

where x is a declared variable that receives the decimal place value of the continuous input or output.

y is an alias or address for a ControlBlock continuous input or output.

NOTE: I/O block addresses can be used only in batch versions 14 and above.

x=**schi**(y) Assigns the “Eng Max” of a ControlBlock input or output to a variable.

where x is a declared variable that receives the “Eng Max” of a ControlBlock input or output.

y is an alias or address for a ControlBlock continuous input or output.

NOTE: I/O block addresses can be used only in batch versions 14 and above.

x=**sclike**(y,z) Converts a value from ControlBlock scaling to Plant scaling.

where x is a declared variable that receives the converted value in Plant scaling.

y is an alias or address for the ControlBlock. The ControlBlock specifies the analog conversion scale that is used by *sclike*.

NOTE: I/O block addresses can be used only in batch versions 14 and above.

z is a variable with a value in ControlBlock scaling that is converted to Plant scaling.

x=**sclo**(y) Assigns the “Eng Zero” of a ControlBlock input or output to a variable.

where x is a variable that receives the “Eng Zero” value.

y is an alias or address for a ControlBlock continuous input or output.

NOTE: I/O block addresses can be used only in batch versions 14 and above.

(continued on next page)

RBL Instructions (continued)

send_bid(x$)

Used with *wait_bid* to generate Batch IDs for Control Recipes. *send_bid* sends the Batch ID to the Control Recipe.

where x is the Batch ID sent to the Control Recipe. You can use a character string in quotes or use a string variable for the Batch ID. The Batch ID can have a maximum of 32 characters. A tag mask must be configured for Batch IDs over 8 characters.

a$=**setbit$**(b$,c)

Sets each of up to eight bits in a character byte equal to 1. If the string expression contains more than one character byte, only the first character byte is set.

where a$ is a string variable that receives the character value.

b$ is the string expression. The setbit function sets bits in the first character byte in b$.

0-7 are the bits (0-7) that are set in the specified byte. You can specify one or more bits separated by commas.

sleep(**x**)

Suspends script execution for a specified time interval.

where x is a user-defined time interval, in seconds.

x=**ssm**

Assigns elapsed time (seconds since midnight) to a declared variable.

where x is a declared variable.

x=**sss**

Assigns elapsed time (seconds since Sunday) to a declared variable

where x is a declared variable.

start x

Directs execution of the script to a phase name and clears the nesting stack of local values (global values are retained).

where x is the phase name (for example, ACT1:).

x=**started_by**(y$)

Returns information on how the recipe was started.

where x is a variable that indicates how the recipe was started. Values include:

1 Recipe started by operator.

0 Recipe started by another recipe.

y$ is a string variable that receives the identification of the user of the recipe that starts the recipe. Values include the user key identification and the recipe name.

(continued on next page)

RBL Instructions (continued)

stop

Stops the execution of the script. If stop is included in a unit or formula script, the next script in the task is downloaded and executed. An *on stop* instruction will execute in response to a *stop* instruction.

stop_task(x)

Stops the execution of a task.

where x is the tag name of the Batch Run screen or PeerWay address of a batch task (for example, 7702 or 77:02 for ControlFile 77, task 02).

y=**status_task**("x")

Checks the status of a task.

where x is the Batch Run screen task or PeerWay address of a batch task (for example, 7702 or 77:02 for ControlFile 77, task 02).

where y is a variable that is used to check the status of a specified task. For a complete list of status values, see RB: 1-6.

a$=**str$**(b)

Converts a floating point number to an ASCII numeric string.

where a$ is a string variable that receives the ASCII numeric string.

b is the floating point value to be converted to a string.

a=**strcat$**(b$,c$)

Returns a character string that is the result of concatenating a second string to the end of a first string.

where a$ is a string variable that receives the concatenated string.

b$ is the first string expression.

c$ is the second string expression.

rs=**strcmp**(b$,c$)

Compares two character string expressions and returns a value indicating whether the expressions are equivalent.

where rs is a status variable. Values include:

1 Character strings do not match.

0 Character strings are equivalent.

b$ is the first string expression.

c$ is the second string expression.

(continued on next page)

RBL Instructions (continued)

string variable	A user-defined name for a character string variable.
syntax **string** x$	Parts of a string variable can be accessed by specifying a character index (for example, a$(1,5)). where x$ is the name of the string variable. String variable names must end in a dollar sign ($).
a=**strlen**(b$)	Returns the character count, in bytes, of a string expression. Each character in a string is represented by one byte. where a is a variable that receives the character count in bytes. b$ is the string expression.
x$=**swab$**(y$)	Swaps and transposes two ASCII bytes when passing the byte values from one string variable to another. where x$ is the string variable that receives the swapped bytes. y$ is the string variable that passes the two bytes.
sync(x)	Synchronizes the execution speed of a while loop. where x is the desired execution speed of the *while* loop.

(continued on next page)

RBL Instructions (continued)

x=**time**(n) Assigns the time of day (time n) to a declared variable.

where x is a declared variable.

n is one of the following time values:

0 Minutes since 1/1/1980

1 Seconds

2 Minutes

3 Day of the Month

5 Month of the year

6 Year

7 Day of week

15 Days in month

16 Days in year

21 Time (HHMMSS)

22 Date (YYMMDD)

23 Date (MMDDYY)

30 System time in hundredths of seconds since CP last started

(continued on next page)

RBL Instructions (continued)

rs=**unacquire**(x)

Terminates master task ownership of a slave task and directs processing of the slave task to an *on unacquire* trap.

where rs is a status variable. Values include:

1 Instruction was successful.

0 Instruction was unsuccessful.

x is the tag of the slave task to unacquire.

undisplay("x",y)

Removes the *display* instruction variable and text designation.

where x is a line of text to be removed from display on the batch screen.

y is a variable to be removed from display on the batch screen.

x=**unsc**(y,z)

Converts a value from Plant scaling to ControlBlock scaling.

where x is a declared variable that receives the converted value in ControlBlock scaling.

y is an alias or block address for the ControlBlock. The ControlBlock specifies the analog conversion scale that is used by *unsc*.

NOTE: I/O block addresses can be used only in versions 14 and above.

z is a variable that contains a value in Plant scaling to be converted to ControlBlock scaling.

until expression

<instructions>

Evaluates expression:

- True – Do not execute the indented instructions.
- False – Execute the indented instructions, then evaluate the expression again.

Example: until TOT1 > TARGET
<instructions>

(continued on next page)

RBL Instructions (continued)

usegraphic("x","y",z) Assigns a process graphic file to the recipe. After the *usegraphic* instruction has executed, you can call up the graphic by selecting the "Graphic" field, the [ACTIVE ALARM] key, or the recipe task on the Batch Overview screen.

where x is the disk volume on which the graphic is located.

y is the name of the graphic file.

z is the number of an anchor object on the process graphic. If there is no anchor on the process graphic, enter 0 in this field.

a=**val**(b$,c) Converts numeric character in a character string to a floating point number, and returns an index to the first non-numeric character after the numeric character.

where a is a variable that receives the index number. A "0" indicates that the conversion was unsuccessful.

b$ is the string variable containing the numeric character to be converted to a floating point number.

c is the floating point variable that receives the floating point value.

rs=**varray_file**("f",x) Changes files that are used by a virtual array (vdim or vstringdim). After *varray_file* changes the file, the virtual array reads and writes to the new file.

where rs is a status variable. Values include:

0 Instruction successfully changed files.

1 Dimensions of the new file are undefined or 0.

2 The new file is the wrong file type.

3 The new file (*vstringdim* only) is the wrong file size.

4 The new file (*vstringdim* only) is the wrong version.

f is the name of the new virtual array file. File names can be a character string or a string variable.

x is the name of the virtual array.

(continued on next page)

RBL Instructions (continued)

virtual array
syntax
rvdim w(x,y) file
rvstringdim w$(x,y,z) f
vdim w(x,y): file
vstringdim w$(x,y,z) f

A user-defined name for a virtual array of data values that are stored in an SRU data file. In a one-dimensional array, each storage location is identified by a single index; in a two dimensional array, each storage location is identified by a row index and a column index.

NOTE: Rules for using virtual arrays:

- File space is not actually assigned to a storage location until a value is written to it. This enables very large arrays to be declared.
- A child script that is linked by a *chain* instruction cannot redefine a virtual array file. If this is attempted, the system assigns "$$default" as the name of the redefined virtual array file in the child script.
- Redundant virtual arrays (rvdim and rvstringdim) save redundant copies of the array file on primary and backup volumes, which are defined on the Batch Configuration screen.

where w is the array name.

x is the number of storage locations in a one-dimensional array or rows in a two-dimensional array. For example, x=40 creates 40 storage locations or rows indexed 0 to 39.

y is the number of columns in a two-dimensional array. For example, y=40 creates 40 columns that are indexed 0 to 39.

z is the maximum number of characters in each character string. Character strings can have up to 269 characters.

f is the name of a file in the SRU Data Folder. It is required when the array is first defined, but not in subsequent declarations.

a=**vfree**

Assigns a value for available volatile memory to a variable.

where a is a variable that receives the volatile memory value.

(continued on next page)

RBL Instructions (continued)

wait_bid(x,y,z)

Used with *send_bid* to generate Batch IDs for Control Recipes. *wait_bid* waits for a request from a Control Recipe for a Batch ID. The *send_bid* instruction sends the Batch ID.

where x is the node number of the CPIV that runs the recipe.

y is the number of the batch task (1 to 32), which is assigned to the recipe on the Batch Units Table.

z is the length of the Batch ID, which is defined on the Batch Configuration screen.

while expression

<instructions>

Evaluates expression:

- True – Execute indented instructions, and then evaluate expression again.
- False – Do not execute indented instructions.

Example: while TOT1<=TARGET

<instructions>

v=**whois**(w,x$,y$,z)

Returns password information about a user at the selected console node: login name, user name, and permission level.

where v is a variable used to check instruction status.

w is the node number of the console. If you enter 0, you can imbed the *whois* instruction under a *prompt* instruction to return information about the user who responds to the prompt.

x$ is a string variable that receives the login name of the user at the console.

y$ is a string variable that receives the user name of the user at the console.

z is a variable that receives the permission Identification number as a binary decimal for the user at the console.

Formatted Print Output Options

To precisely control the formatting of printed output for reports, screen messages, or log entries, you can include print modifier options in a character string message.

Format Control Options

Modify format of printed strings.

Example:	print(*," ^n ^t ^1r print message")
^	Form control designator
^n	Printing to begin on new line
^t	Horizontal tab (8 columns per tab)
^1 x	Color of print statement: 1n, 1r, etc.

Constant Conversion Options

Convert constant values that are system defined into character string output. The % designator option marks where the converted value will appear in the string.

Example:	print(*,"Output printed on %DATE")
%	String Format Designator
%DATE	Current date
%FILE	File name
%ID	Batch ID string
%MAT	Material name of M-Icon
% NODE	Current batch node number
%OPER	Operation name of O-Icon
%RTAG	Working Recipe Tag
%SCRIPT	Script name
%TAG	Batch tag
%TASK	Current batch task number
%TIME	Current time
%VOL	Script Volume

Decimal Conversion Options

Designate format of character string decimals.

Syntax:	print(“% -0nn.dd.x”,y)
Example:	print(1,“^n Left justify %-4.3f”,MAX) print(1,“^n Right justify %04.3f”,MAX)
-	Left justification of string (default is right justified)
0	Pad unused spaces in field width with zeros
nn	Minimum field width to the left of decimal
.dd	Decimal places for floating point number
x	Variable Conversion option
y	Variable

Variable Conversion Options

Convert variable values into character string output. There must be one % designator option for each variable that is converted in the string (in the example, VALUE and A$ are variables). The % designator option marks where the converted value will appear in the string.

Example:	print(*****,*“error number: %d %s”*, *VALUE*, *A$*)
%d	Decimal signed integer
%u	Decimal unsigned integer
%o	Octal unsigned integer
%b	Binary unsigned integer
%x	Hexadecimal unsigned integer
%e	Floating point number in exponential notation
%f	Floating point number
%r	Realtime for date and time-of-day
%rt	Realtime for time-of-day
%rd	Realtime for date
%s	Character string

System Strings

System strings re strings whose values are already defined. The values can be used in *print* instructions to print particular types of data. They are similar to constant conversion options except that they cannot be included directly within the character string. A % designator option must mark where the converted value will appear in the string:

Example:	print(*,“Today is %s”,DATE$)
BACKUP$	Backup volume that is specified on the Batch Configuration screen
DATE$	Today’s date where format is: DDMMYY
FILE$	RBL file name
ID$	Current task ID
MASTER$	Disk volume name
MAT$	Material name of M-Icon
NODE$	Current batch node number
OPER$	Operation name of O-Icon
RTAG$	Working Recipe tag
SCRIPT$	Name of the script
TAG$	Tag of batch task
TASK$	Current batch task number
TIME$	Current time

On Traps

The *on* instruction sets a trap to direct execution of the script to the *on* instruction when a specified event occurs. There are two types of *on* traps:

Alarm Traps

Direct execution in response to an alarm condition or an event that is specified by a ControlBlock logic step or an analog alarm. For, example:

```
alias THIS_EVENT:V105/a
on THIS_EVENT
 <instructions>
 return
```

System Traps

Direct execution of the script to system traps in response to errors in the batch hardware and software. System *on* traps include:

on abort	on nest_err
on acquire_err	on no backup
on chain_err	on power_up
on file_n_err	on stop
on input_err	on task_retry
on link_n_err	on unacquire

The *on* trap instructions are subject to the following rules:

- Global *on* traps are always active during the execution of the script.
- Local *on* traps within a phase are only active during the execution of the phase or during execution of a phase connected by a *gosub* instruction.
- An *on* alarm trap only executes when the alarm or event occurs. If the alarm condition is already in effect when the *on* instruction is activated, the *on* trap does not recognize the alarm and will not execute.
- The *on* traps may be active in more than one phase if these phases are connected by *gosub* instructions. If a *gosub* instruction is used to direct execution to another phase, on traps in the calling phase, the called phase, and the global environment are all active.
- If there is more than one active *on* trap for the same ControlBlock address or system error, the *on* trap in the local or lowest nesting level will have execution priority over the other *on* traps.
- A *return* instruction returns execution back to the line that was executing before the *on* instruction was executed.
- While scripts are linked by *chain* instructions, only the *on* traps in the child script are active. None of the traps in the parent script are active until the child script executes an *endchain* instruction to return execution to the parent script.
- An alarm *on* trap will not execute unless the “Report” and “When” fields on the ControlBlock discrete diagram display are configured. These fields must be configured for the ControlBlock to generate an alarm.
- If an *on* trap is executing when another alarm or system error occurs for another active *on* trap, the executing *on* trap is interrupted and the next *on* trap is executed. When the next *on* trap finishes executing, the first *on* trap resumes execution. Execution can jump from one *on* trap to another for a maximum of 40 nesting levels.
- A *start* instruction clears the *on* traps in all nesting levels. If an *on* trap executes a *start* instruction, all other traps that have been interrupted will not finish.

Controlling *while*, *until* and *for* Loops with *sleep*

RBL *while*, *until*, and *for* loops that execute repeatedly in quick succession can overwhelm the PeerWay with messages. To slow down the execution of the loop, include a *sleep* instruction in the loop as follows.

```
while (alias)
   sleep(2)
```

Without a *sleep* instruction to control loop speed, the loop can execute faster than the controller can update data, wasting ControlFile resources.

Because the appropriate amount of sleep time varies with the application, there is no easy way to specify the amount sleep time for every application. As a general rule, use the maximum amount of sleep time that your application can accommodate. Typical sleep times range from 1 to 10 seconds.

Mathematical Operators

Operators are used for data manipulation.

- Logical operators indicate and test for alternative values.
- Relational operators test for equality of values.
- Arithmetic operators perform arithmetic operations.

Logical Operators

Operator	Description
v=x?y:z	if-else selection — If x is true, v equals y; if x is false, v equals z.
x&y	logical “and” — True if both expressions are true; false otherwise.
x|y	logical “or” — True if one or both expressions are true; false otherwise.
x↑y	logical “exclusive or” — True if only one expression is true; false otherwise.

Relational Operators

Operator	Description
x~=y	relational inequality
x>=y	relational greater than or equal to
x>y	relational greater than
x<=y	relational less than or equal to
x<y	relational less than
x==y	relational equal
(true)	Value is > 0; returns 1
(false)	Value is <= 0; returns 0

Arithmetic Operators

Operator	Description
exp2 x	exponentiation, 2 to the power of x
log x	base 10 log of x
ln x	base e log (natural log) of x
exp x	exponentiation, e to the power of x
x**y	exponentiation, x to the power of y
x*y	multiply x times y
x/y	divide x by y
x%y	modulo remainder of x/y
x max y	maximum value of x and y
x min y	minimum value of x and y
x+y	add x and y
~x	unary logical negation of x
( )	parenthesis for grouping
round x	round x to the nearest integer
int x	truncate x to the lesser integer
fract x	return the fractional part of x
abs x	absolute value of x
sign x	signee (sign) returns +1, 0, or –1
sqrt x	square root of x
sqrl x	square root limited gain of x

(continued on next page)

Arithmetic Operators (continued)

Operator	Description
sin x	sine trigonometric function of x (radians)
asin x	inverse sine of x (radians)
cos x	cosine of x (radians)
acos x	inverse cosine of x (radians)
tan x	tangent of x (radians)
atan x	inverse tangent of x (radians)
log2 x	base 2 log of x

Section 7:
System Limits

Values and Limits

This section lists ABC Batch and Rosemount Basic Language values and their corresponding limits.

NOTE: Some values are affected by available volatile and nonvolatile memory.

Values and Limits

Value	Limits
Aliases	Amount of symbol space
Arrays (dim)	250 arrays per task 255 indices per array dimension
Arrays (vdim)	50 x 50 kilobytes
Batch units per batch unit set	16
Batch units per Batch Units Table	256
Batch unit sets per unit sec file	30
Branches per Decision icon	16
Characters per branch name of a Decision icon	16
Characters per alarm message	21
Character per Batch ID	32
Characters per icon comment	60
Characters per icon name	10
Characters per material description	35
Characters per material property name	10
Characters per parameter description	35
Characters per parameter name	18
Characters per recipe label	10

Values and Limits

Value	Limits
Characters per recipe tag	8
Characters per string variable	269
Characters per variable name	18
Constants	Amount of symbol space
Entries per Batch Formulas Table	32,000
Icons per recipe	98
Global variables	250 per batch task
Local variables	250 per batch task
Materials per Batch Materials Table	1,024
Material properties per material	20
Non Volatile memory	14.8 Kilobytes (CP2) 56 Kilobytes (CP4) 32 Kilobytes (SRU)
Open report files	10
operations per Batch Operations Table	1,200
Operations per material	10
Operations per recipe	1,000
Parallel icons per recipe row	16
Parameters per Batch Operations Table	18,000
Parameters per operation	1,000
Parameters per recipe	1,024
Private dim arrays per script	250
Private local variables per script	250
RBL files	1000
Scripts per operation	10

Values and Limits

Value	Limits
RPN space	31000 bytes
Script	10000 lines, 31000 bytes
Scripts in a file	100 scripts
String variables	250 strings per batch task
Symbol space	16000 bytes per batch task
Tasks	64 tasks
Volatile memory	200 Kilobytes (CP2) 786 Kilobytes (CP4) 500 Kilobytes (SRU)

FISHER-ROSEMOUNT

RS3™
ABC Batch Quick Reference Guide

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W