Honeywell

Experion PKS

Batch Overview and Planning Guide

EPDOC-XXX7-en-431A
February 2015

Release 431

Honeywell

Document	Release	Issue	Date
EPDOC-XXX7-en-431A	431	0	February 2015

Disclaimer

Contents

1 About This Guide 5

1.1 Basic concepts and terms 6

2 Batch control concepts 13

2.1 Overview of S88 physical model 15

2.2 Overview of S88 Procedural (Recipe) model 16

2.3 Linkage between S88 physical model, S88 recipe model, and process model 17

3 Overview of Experion Batch Manager 19

3.1 Support for S88 - Benefits in EBM 20

3.2 EBM Recipes 21

3.2.1 Layered recipe model - Mapping to S88 21

3.2.2 Phase blocks serve as layer links 22

3.2.3 About using Data blocks and passing formula and report parameters between recipe layers 22

3.2.4 Instance-based recipes 24

3.2.5 Class-based recipes 25

3.2.6 About unit acquisition and passing 26

3.3 Overview of Procedural Operations 28

4 Overview of EBM Equipment model 29

4.1 About building Phase block types 31

4.2 About Unit classes 32

4.3 About Map block types 33

4.4 About Unit instances/UCMs 34

5 Build-time support - Engineering tools 35

6 Overview of Activities 37

6.1 Activities-related Operational UIs 38

7 Batch Application Services 41

8 Monitoring, diagnostics, and post-execution support 43

9 Overview of intercluster peer-to-peer communication 45

10 General planning guidelines 47

10.1 Size of a project 48

10.2 Licensing considerations 49

10.3 Distribution across multiple Control Execution Environments 50

10.4 Scope of Responsibility considerations 51

10.5 About enabling Formulators to access Recipe Builder 52

10.6 About setting permission levels for using Recipe Builder 53

11 Guidelines/considerations for building equipment model 55

11.1 Formula or Report data characteristics 56

11.2 Considerations for Phase block types 57

11.3 Considerations for enumeration sets 58

11.4 Differences between a UCM and a Unit class 59

11.5 Usage of Map blocks 60

11.5.1 Considerations for creating functions 60

11.5.2 Considerations for creating block references 60
11.6 Considerations for modifying an equipment model 62
11.6.1 Restrictions for modifying enumeration sets 62
11.6.2 Considerations for modifying Phase block types 62
11.6.3 Considerations for modifying Map block types 62
11.6.4 Considerations for modifying Unit classes and UCMs 63

12 Guidelines/considerations for building recipes 65
12.1 Differences between instance-based recipes and Class-based recipes 66
12.2 Acquisition of resources support 67
12.2.1 Acquisition support in EBM 67
12.2.2 Parent and child recipes 67
12.3 Alias table and user-defined templates 69
12.4 UCM or equipment name versus alias reference usage 71
12.5 Memory utilization and loading considerations 72
12.6 Using an existing RCM or an SCM in Class-based recipes 73
12.7 Using an RCM or SCM configured to use HISTVALUE and RECTARGET as a child of a Master Recipe/Control Recipe 76

13 Guidelines/considerations for building custom graphics for Class-based recipes 77

14 About import/export support for Class-based recipes 79
14.1 About import/export of Map block types 82
14.2 About import/export of Unit classes 83
14.3 About import/export of Proxy strategies 84

15 Planning for intercluster peer-to-peer 87
15.1 Distribution of recipe layers across clusters 88
15.2 Guidelines for configuring intercluster peer-to-peer communication 89
15.3 Considerations for Phase block type mastership configuration 90
15.4 Considerations for selecting communication ranges for controllers 91

16 Notices 93
16.1 Documentation feedback 94
16.2 How to report a security vulnerability 95
16.3 Support 96
16.4 Training classes 97

1 About This Guide

This guide provides information to assist you in planning the batch related activities.

Revision history

Revision	Date	Description
A	February 2015	Initial release of document.

1.1 Basic concepts and terms

The following table defines some common industry and Honeywell-specific terms that are associated with sequential, procedure, and batch control.

Attention
Honeywell-specific terms are identified with an asterisk (*)

Term/Abbreviation	Description
Allocation	A form of coordination control that assigns a resource to a batch or unit. Note that an allocation can be for the entire resource or for portions of a resource.
Alternative path	Part of a Recipe Procedure, where only one of the alternative step sequences is activated.
Application Control Environment/ACE*	Experion Level 2/3 supervisory control platform that executes on a Windows platform not within a hardened purpose built controller, such as C300.
Arbitration	A form of coordination control that determines how a resource should be allocated when there are more requests for the resource than can be accommodated at one time.
Asset hierarchy	A concept in Enterprise Model (similar to Plant Data Model) to define the containment-based hierarchical model of equipment entities and unit equipments.
Basic control	Control that is dedicated to establishing and maintaining a specific state of equipment or process condition. It may include regulatory control, interlocking, monitoring, exception handling, and discrete or sequential control.
Batch	In this document, a batch means an entity that represents the production of a material at any point in the process.
Batch control	Control activities and control functions that provide a means to process finite quantities of input materials by subjecting them to an ordered set of processing activities over a finite period of time using one or more pieces of equipment.
Batch ID	A string to identify a Control Recipe. Batch ID is not critical to a Master Recipe.
Batch process	A process that leads to the production of finite quantities of material by subjecting quantities of input materials to an ordered set of processing activities over a finite period of time using one or more pieces of equipment.
Central repositories*	Experion system location for storage of the equipment hierarchy, class information, unit tags, recipe parameters, formula parameters, etc. This may be a combination of applications such as pieces stored as types in Control Builder and other pieces stored in the Enterprise Model Builder (EMB).
Child recipe	In a recipe that has two or more layers, the child recipe is at lower level, and is controlled by the recipe above it. The parent/child concept in layered recipe is not associated with parent/child relationship in user-defined template function
Class-based recipe	Class-based recipes function allows you to build a single recipe against a group of similar units, and instantiate multiple instances at run-time. These instances can be executed simultaneously against different units. The recipe is built once, tested once, and then run as many instances as needed (within the system limits). Without Class-based recipes function, you have to build a recipe for every single unit instance and maintain a large number of recipes. Each recipe instance must be tested individually, so the testing effort could be significant. Class-based recipes function helps reduce the cost on recipe engineering, maintenance, and testing.

Term/Abbreviation	Description
Cluster block*	A tag block that represents a local cluster for establishing intercluster peer-to-peer communication.
Cluster ID*	The unique identifier configured (either automatically or manually) for each cluster involved in intercluster peer-to-peer.
Common device Control Modules*	A Control Module (CM) configured to act as resource to be controlled and acquired by SCMs and *Recipes*.
Common resource	A resource that can provide services to more than one requester. Note that common resources are identified as either exclusive-use resources or shared-use resources.
Control Module	The lowest level grouping of equipment in the physical model that can carry out basic control. This term applies to both the physical equipment and the equipment entity.
Control Recipe	A type of recipe which, through its execution, defines the manufacture of a single batch of a specific product. A Control Recipe is the result of loading a Master Recipe and resides in the controller (real-time). Control Recipes are instantiated from Master Recipes and they are the executable version of the Master Recipes. A Control Recipe is associated with a specific Unit instance selected by the operator/program, and executes against that Unit instance at run-time.
Coordination control	A type of control that directs, initiates, and/or modifies the execution of procedural control and the utilization of equipment entities.
Data block*	A non-executable basic block that contains custom parameters. It is created from Phase block type, and contained by recipes or SCMs. Data block is typically used to carry recipe level formula and report parameters, and it is a means to automatically align formula and report parameters between the Phase block and its underlying recipe or SCM (by using the same data structure defined in the Phase/Data block).
Enterprise	An organization that coordinates the operation of one or more sites.
Equipment control	The equipment-specific functionality that provides the actual control capability for an equipment entity, including procedural, basic, and coordination control, and that is not part of the recipe.
Equipment module	A functional group of equipment that can carry out a finite number of specific minor processing activities.
Equipment operation	An operation that is part of equipment control.
Equipment phase*	An SCM created to implement a phase to control a specific equipment entity. An equipment phase can be implemented also with a Unit Control Module (UCM) that is linked to a Control Recipe.
Equipment procedure	A procedure that is part of equipment control.
Equipment unit procedure	A unit procedure that is part of equipment control.
Exception handling	Those functions that deal with plant or process contingencies and other events which occur outside the normal or desired behavior of batch control.
Exclusive-use resource	A common resource that only one user can use at any given time.
Food and Drug Administration/FDA	Responsible for compliance with pharmaceutical regulation in and for the United States. For more information, refer to https://www.honeywellprocess.com/library/support/Documents/Customer/Series_C_IO_Users_Guide_EPDOC-X126-en-430.pdf.
Formula	In ISA-S88.01 terms, a category of recipe information that includes process inputs, process parameters, and process outputs. In Honeywell terms, the preferred/recommended implementation of formula is by using Phase block formula and report parameters.

Term/Abbreviation	Description
Formula parameter definition*	A set of user defined variables used in a Phase block to communicate with the underlying SCM or *Recipe*.
Functional asset model	A model representing the "Functions" performed within a Batch manufacturing facility. The functions are clearly defined and define the "contract" between the requesters of the function and the providers, which allows a single Recipe to execute different actual process actions (such as SCMA on AssetA, SCMB on AssetB, RCMD on AssetD) against different Assets, while providing a single consistent view from the requestor (such as FEED).
Handler*	The execution engine for a *Recipe* or a Sequential Control Module.
Header	The administrative information in the recipe is referred to as the header. Typical header information may include the recipe and product identification, the version number, the originator, the issue date, approvals, status, and other administrative information.
The International Electrotechnical commission/IEC	The international standards and conformity assessment body for all fields of electrotechnology. The American National Standards Institute (ANSI) is a member of IEC. For more information, refer to http://www.iec.ch/.
IEC output*	A step output in SCM or RCM of type S_IEC, N_IEC or R_IEC. These outputs are used to control parameters in Common Device Control Modules.
IEC exit processing*	The output processing which takes place when a step is about to be deactivated and the next step is becoming the active step. This is an internal concept that is not visible to the user. Typical actions are: resetting the destination value to OFF (for N_IEC outputs) and releasing the device, if required.
The International Society of Automation/ISA	Mission is to maximize the effectiveness of ISA members and other practitioners and organizations worldwide to advance and apply the science, technology, and allied arts of instrumentation, systems, and automation in all industries and applications. For more information, refer to http://www.isa.org/.
Intercluster peer-to-peer*	Communication between two controllers in separate Experion clusters. Implies that the configuration for each controller involved is in a separate ERDB.
Layered recipe	Multiple recipes that are linked together in a hierarchical fashion (procedure, unit procedure, operation and phase). The recipe at higher level in the hierarchy controls the one(s) below it. The execution of layered recipe manages the acquisition of unit and equipment, formula and report parameter passing between recipe layers, state tracking and propagation between recipe layers, mode attribute handling etc, in addition to managing execution of the sequences.
Linked phase*	A phase that is implemented with a Phase block.
Map block*	Map block types define functions and references that can be contained by Unit classes/Unit.
Master Recipe	A type of recipe that accounts for equipment capabilities and may include process cell-specific information. Class-based Master Recipes are built against Unit classes. They serve as "recipe templates."
Mode	The manner in which the transition of sequential functions are carried out within a procedural element or the accessibility for manipulating the states of equipment entities manually or by other types of control.

Term/Abbreviation	Description
Null Phase block*	Null Phase block is a special configured Phase block which doesn't control any recipe or SCM. Null Phase block contains custom parameters. It has the ability to store formula parameters to their destination, and retrieve report data from the source of the report parameters. Null Phase block is normally used to carry recipe level local formula and report parameters. It can also be used in parallel execution in a recipe sequence to create delay.
Operation	A procedural element defining an independent processing activity consisting of the algorithm necessary for the initiation, organization, and control of phases.
Parent recipe	One of the recipes in two consecutive layers in layered recipes. Parent recipe is the one at upper level, and controls the recipe below it. The Parent/Child concept in layered recipe is completely different from Parent/Child relationship in User Defined Template function.
Phase	The lowest level of procedural element in the procedural control model.
Phase block	Phase block is a custom block type used for defining formula and report parameters. It defines the interface between recipe layers.
Phase level contract formula parameter	Formula data structure owned by a Phase. It is the contract between the Phase block and its underlying *Recipe*/SCM, therefore it is the same data structure owned by the underlying *Recipe*/SCM. The data in this structure gets passed to the underlying recipes/SCMs when the Phase is executed. It is a subset of its parent formula data.
Phase level contract report parameter	Report data structure owned by a Phase block. It is the contract between the Phase block and its underlying *Recipe*/SCM, therefore it is the same data structure owned by the underlying *Recipe*/SCM. It stores report data coming from underlying *Recipes*/SCMs, and it is a subset of its parent report data.
Physical asset model	A model representing the Physical Equipment upon which the control strategies are executed. This model defined to the level of granularity needed to support aspects of Modular Batch Automation that support the definition of control strategies at a logical functional level for subsequent execution against the defined Assets.
Procedural control	Control that directs equipment-oriented actions to take place in an ordered sequence to carry out some process-oriented task.
Procedural element	A building block for procedural control that is defined by the procedural control model.
Procedure	The strategy for carrying out a process. In general, it refers to the strategy for making a batch within a process cell. It may also refer to a process that does not result in the production of product, such as a clean-in-place procedure.
Procedure Analyst*	Honeywell's product that provides the ability to do post batch execution analysis, and generate batch reports.
Process	A sequence of chemical, physical, or biological activities for the conversion, transport, or storage of material or energy.
Process action	Minor processing activities that are combined to make up a process operation. Process actions are the lowest level of processing activity within the process model.
Process operation	Starting, controlling, and ending a process or procedure.
Procedure function chart	A chart of transition conditions and phases to define the execution order in Recipe Procedure.
Process cell	A logical grouping of equipment that includes the equipment required for production of one or more batches. It defines the span of logical control of one set of process equipment within an area. This term applies to both the physical equipment and the equipment entity.

Term/Abbreviation	Description
Process control	The control activity that includes the control functions needed to provide sequential, regulatory, and discrete control and to gather and display data.
Process input	The identification and quantity of a raw material or other resource required to make a product.
Process output	An identification and quantity of material or energy expected to result from one execution of a Control Recipe.
Process parameter	Information that is needed to manufacture a material but does not fall into the classification of process input or process output. Examples of process parameter information are temperature, pressure, and time.
Process stage	A part of a process that usually operates independently from other process stages and that usually results in a planned sequence of chemical or physical changes in the material being processed.
Proxy node*	A node that exists for configuration and handle-to-name resolution of a node existing in another cluster. Acts as a stand-in (proxy) for configuring peer references to a controller in a separate cluster. It is contained by a cluster block.
Recipe*	Refers to both instance-based recipes (RCMs) and Class-based recipes.
Recipe level contract formula parameter	Formula data structure owned by a recipe. It is the contract between the recipe and the Phase controlling it (if the recipe is not at top level), therefore it is the same data structure owned by the controlling Phase block. It is passed to the contained Phases.
Recipe level contract report parameter	Report data structure owned by a recipe. It is the contract between the recipe and the Phase controlling it (if the recipe is not at top level), therefore it is the same data structure owned by the controlling Phase block. It stores report data coming from the child recipes/SCMs.
Recipe level local formula parameter	Formula data structure used by recipe sequence. It does not get passed to the contained phases.
Recipe level local report parameter:	Report data structure used by recipe to store data computed by its sequence.
Recipe management	The control activity that includes the control functions needed to create, store, and maintain general, site, and Master Recipes.
Recipe operation	An operation that is part of a recipe procedure in a Master Recipe or a Control Recipe.
Recipe phase	A phase that is part of a recipe procedure in a Master Recipe or a Control Recipe.
Recipe Procedure	The part of a recipe that defines the strategy for producing a batch.
Recipe Unit Procedure	A unit procedure that is part of a recipe procedure in a Master Recipe or a Control Recipe.
Requester	Requester performs the acquisition. For example, the requester can be a Control Recipe.
Resource	Resource is being acquired by the requester. For example, the resource can be a UCM, CM, SCM, or *Recipe*.
Shared-use resource	A common resource that can be used by more than one user at a time.
Scope of Responsibility/SOR	A method for identifying functions/operations associated with a given entity.
State	The condition of an equipment entity or of a procedural element at a given time. The number of possible states and their names vary for equipment and for procedural elements.

Term/Abbreviation	Description
Step block	A basic block contained in an SCM or Recipe Procedure. The step block can perform a subset of actions defined in IEC 61131 for sequential control and additional Honeywell-specific actions for the purpose of providing equipment control, interfacing with or performing basic control or helping in the implementation of coordination control. The step block is also a general purpose block for reading and writing values from within any sequence or procedure
TotalPlant Batch	Honeywell's Batch product. Supports ISA S88.01 batch terminology. **Note** TotalPlant Batch (TPB) is a legacy batch product that is not available for new sale. However, Honeywell continues to support the existing TotalPlant Batch (TPB) customer. New batch customers are recommended to buy the Experion Batch Manager (EBM).
Transition condition	A directional link between phases with a dynamic or static condition to control the execution of the Recipe Procedure. The transition condition is always true. In case of alternative paths, a real condition is required for dynamic selection of the active path.
Unit	A collection of associated control modules and/or equipment modules and other process equipment in which one or more major processing activities can be conducted. Other things to consider include the following: • This term applies to both the physical equipment and the equipment control • Examples of major processing activities are react, crystallize, and make a solution.
Unit class	Unit classes are used for organizing plant equipment into classes from which you can build Class-based Master Recipes and instantiate Unit instances. Unit classes provide you the necessary functions and references for building Master Recipes.
Unit instance	Unit instances are strategies instantiated from a Unit class.
Unit Procedure	A strategy for carrying out a contiguous process within a unit. It consists of contiguous operations and the algorithm necessary for the initiation, organization, and control of those operations.
Unit supervision	The control activity that includes control functions needed to supervise the unit and the unit's resources.
Standard	*Description*
IEC 61131-3	Defines automatic execution of Sequential Function charts in programmable controllers.
IEC 61512-1 (Also Known As: ANSI/ISA–88.01–1995)	Here: Provides examples for different operational modes.
IEC 61512-2 (Also Known As: ANSI/ISA–88.00.02–2001)	Here: Defines automatic execution of procedure function charts.

2 Batch control concepts

Batch processes are discontinuous processes. Batch processes are neither discrete nor continuous; however, they have characteristics of both. The product produced by a batch process is called a batch. Batch processes do not involve a constant flow of materials in and out of the process, and the output is normally a homogeneous mass, not discrete objects. After a useful chemical transformation is developed in a laboratory, people reduce the conditions for the transformation to the minimum number required to make the product. These conditions include the amounts of each raw material and the procedures for transforming them. The result is a narrative recipe for making a product. Examples of batch products from a batch process may be dough for bagels, terephthalic acid (TPA) for polyester yarn, the finish solution for permanent press goods, or the base for various shades of paint. The laboratory process may be scaled up to a commercial batch process in which the beakers and Bunsenburners become highly agitated tantalum tanks and multimedia heat exchangers. The following are some of the key characteristics of a batch process.

- Operations occur in sequences followed by a step-by-step set of instructions.
- Production of limited quantity “batch”.
- Environment varies and changes dramatically from one operation to next.

According to the batch standard ISA-88.00.01-CDV20 (ANSI/ISA-88.01-1995, IEC 61512-1), the subdivisions of a batch process can be organized in a hierarchical fashion as illustrated in the following figure.

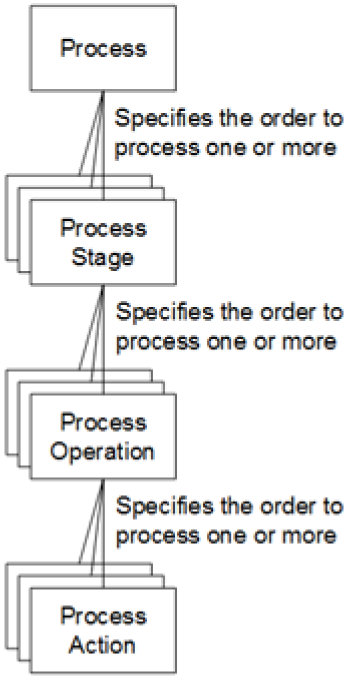

Process
Specifies the order to process one or more
Process Stage
Specifies the order to process one or more
Process Operation
Specifies the order to process one or more
Process Action

Figure 1: Batch process model

Note

Throughout this document, the batch standard ISA-88.00.01-CDV20 (ANSI/ISA-88.01-1995, IEC 61512-1) is referred to as the “S88” model.

Related topics

“Overview of S88 physical model” on page 15
“Overview of S88 Procedural (Recipe) model” on page 16
“Linkage between S88 physical model, S88 recipe model, and process model” on page 17

2.1 Overview of S88 physical model

The physical assets of an enterprise involved in batch manufacturing are usually organized in a hierarchical fashion as described in the following figure.

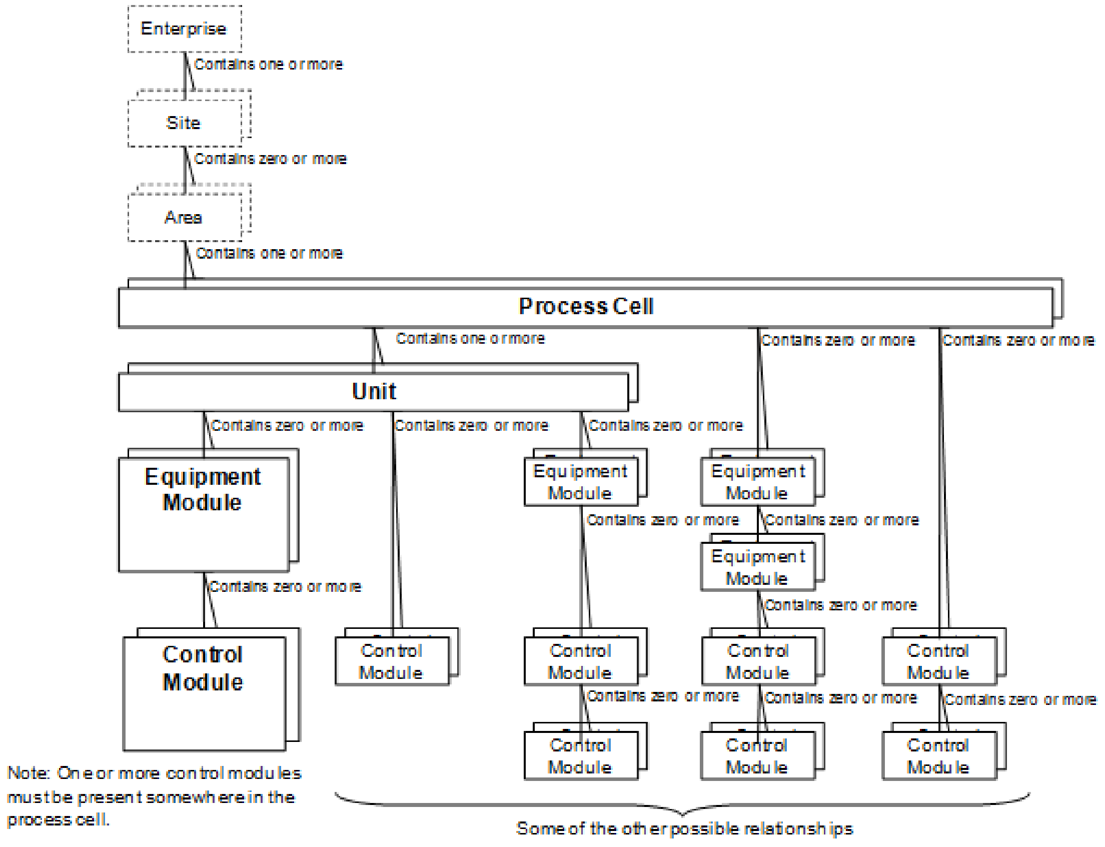


Figure 2: S88 Physical model

The model has seven levels, starting at the top with an enterprise, a site, and an area. These three levels are frequently defined by business considerations and hence, not discussed further. The four lower equipment levels (process cells, units, equipment modules, and control modules) are defined by engineering activities.

2.2 Overview of S88 Procedural (Recipe) model

Procedural control is a characteristic of batch processes. It is the control that enables equipment to perform a batch process. The hierarchy of identified and named procedural elements is illustrated in the following figure and consists of Procedures, Unit procedures, Operations, and Phases.

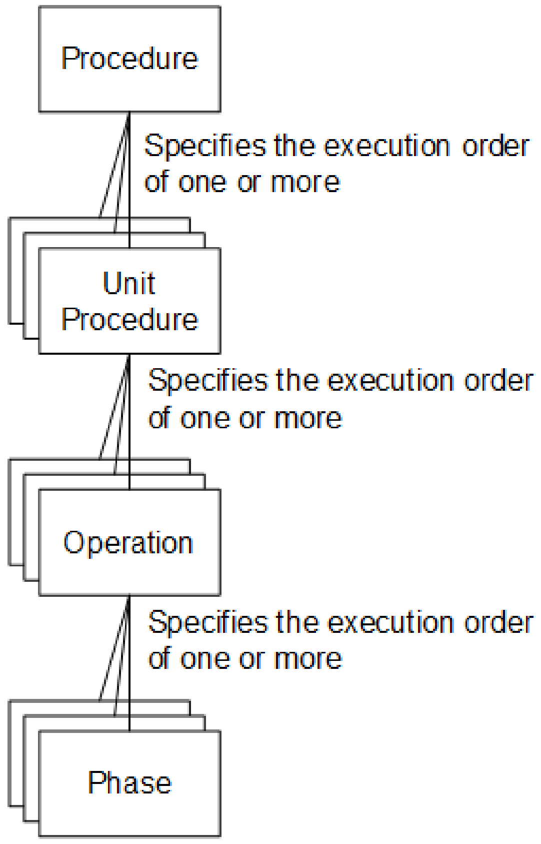

Procedure
Specifies the execution order of one or more
Unit Procedure
Specifies the execution order of one or more
Operation
Specifies the execution order of one or more
Phase

Figure 3: S88 Procedural (Recipe) control model

The Procedure is the highest level in the hierarchy and defines the strategy for carrying out a major processing action such as making a batch.

2.3 Linkage between S88 physical model, S88 recipe model, and process model

The general relationship between the procedural control model, the physical model, and the process model is illustrated the following figure. The procedural control may be entirely defined as part of equipment control, or it may be based on procedural information passed on to the equipment entity from the recipe.

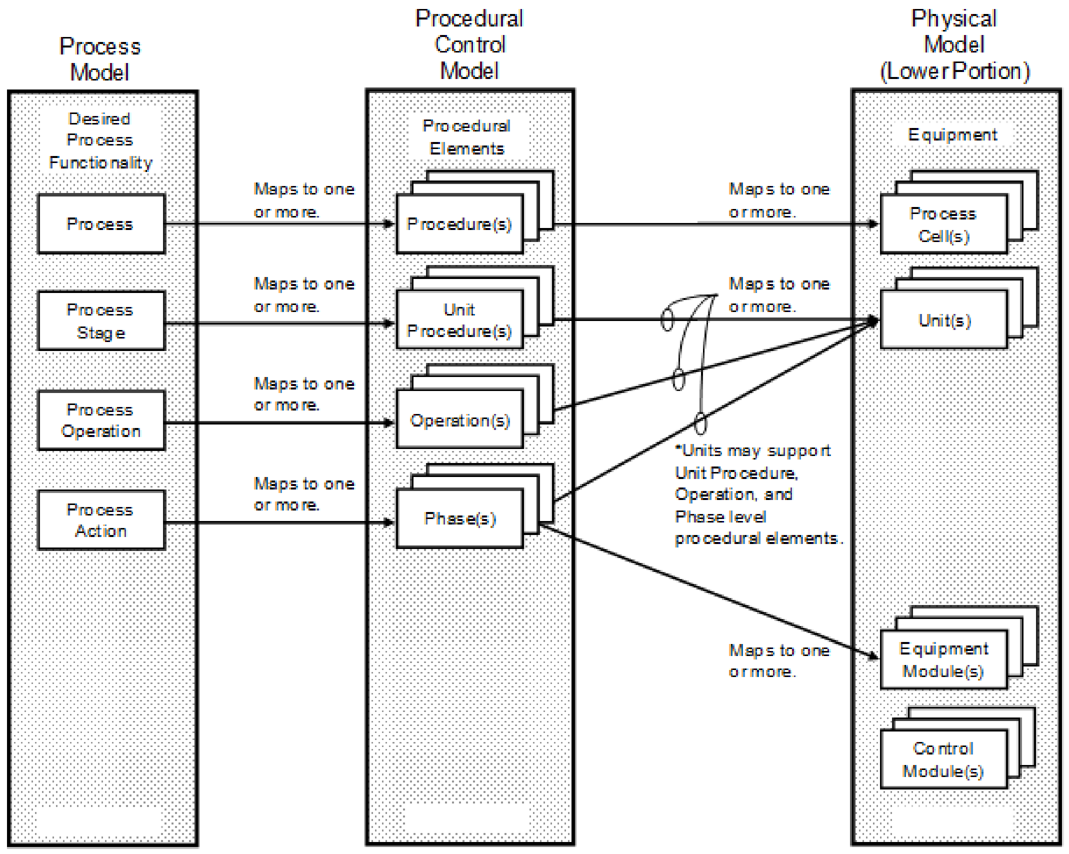


Figure 4: Linkage between S88 Physical, Process, and Procedural model

For more information about the batch control concepts, refer to the *ISA-88.00.01-CDV20, formerly ANSI/ISA–S88.01–1995 Batch Control Part 1: Models and Terminology*.

3 Overview of Experion Batch Manager

Related topics

“Support for S88 - Benefits in EBM” on page 20
“EBM Recipes” on page 21
“Overview of Procedural Operations” on page 28

3.1 Support for S88 - Benefits in EBM

As illustrated in the following figure, the Experion Batch Manager (EBM) layered recipe functionality supports the S88 standards to realize the following major benefits.

- Emphasizes good practices of S88.
- Enables better identification of manufacturing and process needs by developing common functions.
- Promotes a modular approach to automation, which is inherent in Experion systems.

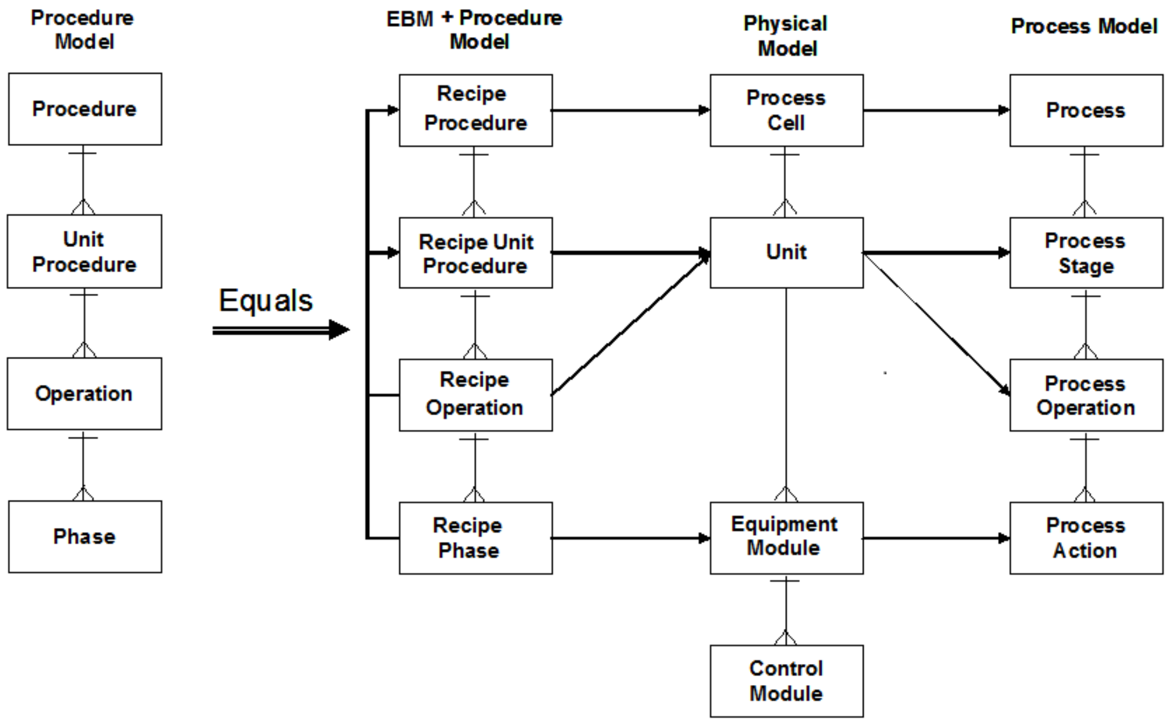


Figure 5: S88 recipe hierarchy and EBM layered recipe

Modular design concept

You can analyze the entire manufacturing operations and develop a library of *functions*. These functions provide the abstraction between parent and child. EBM supports a common level of abstraction by allowing any parent *Recipe* to initiate a child action through either an SCM or a child *Recipe*. This means, you can define the functions, the associated contract to support the function, and how these functions are provided for each cell. This supports the *modular* approach by treating each function as an independent replaceable component and allows components to be distributed throughout the entire system in a manner that meets the performance needs. Modular design allows incremental automation/upgrade for process changes without making large scale changes to a wide range of recipes.

3.2 EBM Recipes

The EBM layered recipe functionality provides the ability to modularize batch plants. It allows you to break an entire batch plant into smaller pieces, and implement, test/validate each piece as an individual module. All pieces (modules) are then integrated to form layered recipes/procedures. This in turn reduces the engineering, validation and maintenance effort.

EBM provides the ability to map a Phase block to a *Recipe*/SCM, which allows a "function" defined by a Phase block to initiate a single simple SCM in one case and a more complex multi-layer *Recipe* in another. This lets layered recipes map directly to an *Recipe*/SCM at any level rather than constraining initiation of a Phase/SCM to the operation level. For example, a Procedure, a Unit Procedure, or an Operation can directly initiate a Phase/SCM. EBM does not constrain to a defined parent/child relationship between the layers. It supports the standard layers while giving the flexibility to meet specific process needs.

3.2.1 Layered recipe model - Mapping to S88

Layered recipe function provides the ability for a recipe to control another recipe or sequence. You can build and execute multiple level hierarchical recipes as defined in S88 model. The following image illustrates how the Experion Recipe model maps to the S88 model.

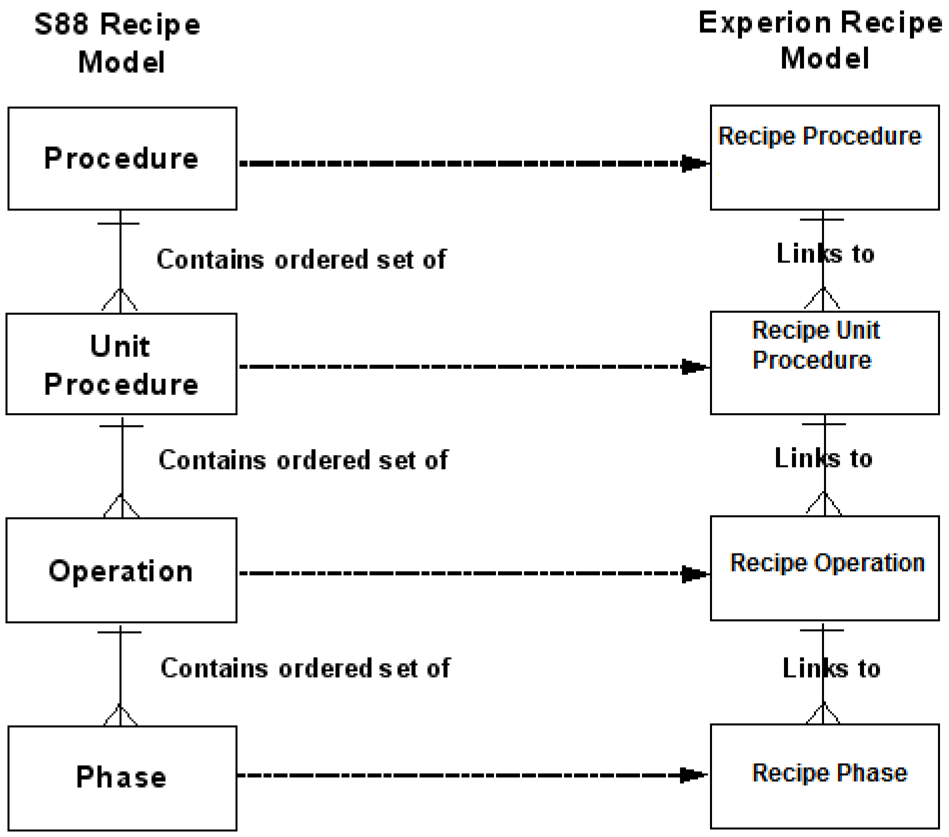


Figure 6: EBM recipe model mapping to S88

The layered recipe function lets you build recipes in layers to support the S88 recipe hierarchy. In this hierarchy, a higher level recipe can control its underlying recipe(s). Recipes at each layer are implemented as a modular function block. Recipe Control Modules (RCMs) or Master Recipes (MRs) can represent *Procedures, Unit Procedures,* and *Operations*. Sequential Control Module (SCM) blocks can represent *Phases*.

3.2.2 Phase blocks serve as layer links

A Phase block type is a custom block type used for defining formula and report parameters. It defines the interface between recipe layers. As an interface, it has the following functions.

- It defines the formula and report parameters of a *Recipe*/SCM. A Phase block type associated with a *Recipe*/SCM is called the Data block of the *Recipe*/SCM and can be accessed by the name “DATA.” (such as, REACT.DATA).
- It is used by *Recipes* to acquire and control child *Recipes*/SCMs by performing the following:
  - (optionally) Retrieves formula values from the *Recipe*
  - Stores formula values to child *Recipe*/SCM Data block at the start of the child execution.
  - (for Master Recipe/Control Recipe) retrieves Unit selection from the Control Recipe at the start of the child Control Recipe execution and passes the Unit selection to the child Control Recipe.
  - Retrieves report values from child *Recipe*/SCM Data block when the child execution is complete.
  - (for Master Recipe/Control Recipe) Retrieves Unit selection from the child Control Recipe when the execution is completed.

Note

The generic term for the usage of a Phase type is a Phase block. Each Phase block within a *Recipe* is assigned a unique name within the scope of the *Recipe*. The Phase types of a Phase block and the Phase types of a Data block of a child *Recipe*/SCM should be of the same type.

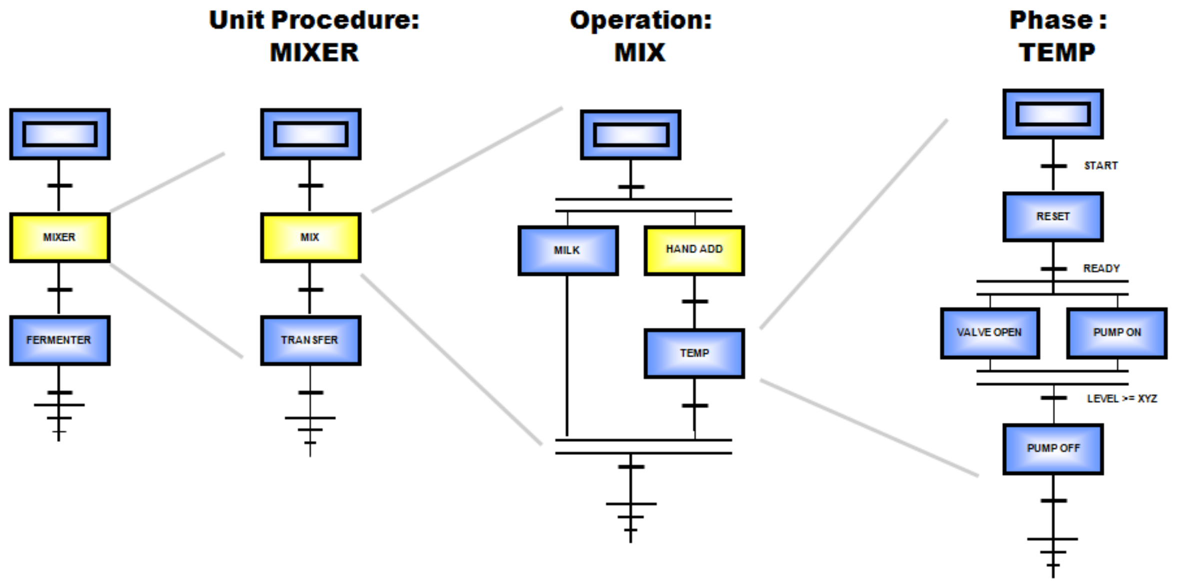


Figure 7: Phase blocks as layer links

3.2.3 About using Data blocks and passing formula and report parameters between recipe layers

The Data block of an SCM/*Recipe* defines the formula and report parameters of an SCM or a *Recipe*. For a top level SCM or *Recipe* without a parent, the Data block defines the formula and report parameters that will be presented to the operator during Batch setup. For a child *Recipe*, these are the formula and report parameters that are passed between the parent *Recipe* and the child *Recipe*. Phase blocks are used for passing formula parameters entered by the operator down to the child *Recipe*/SCM Data block. The Phase block is also used for retrieving the report parameters from the child *Recipe*/SCM Data block.

The following figures illustrate a Data block in a *Recipe*/SCM.

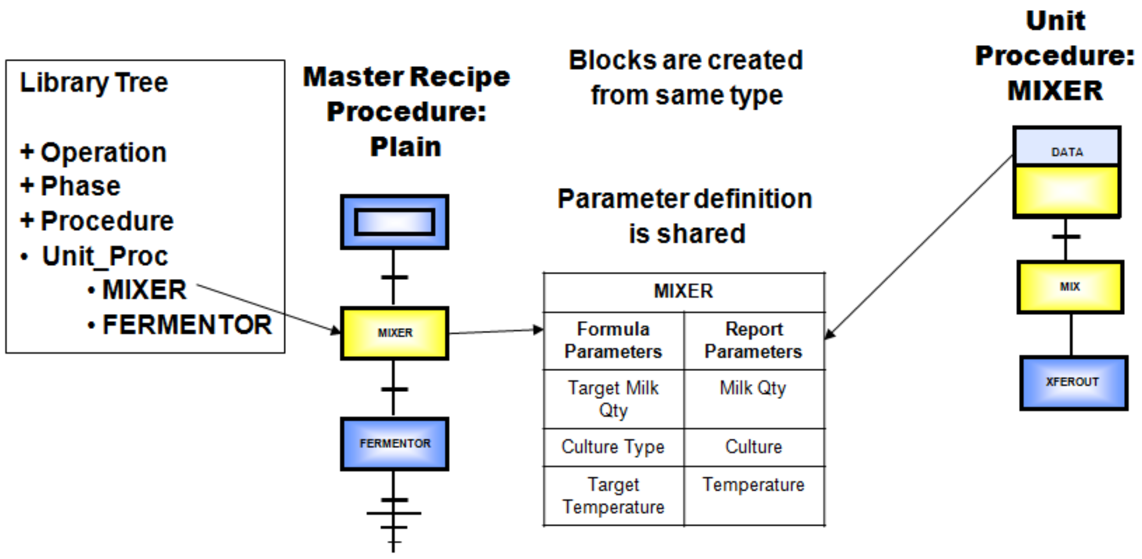
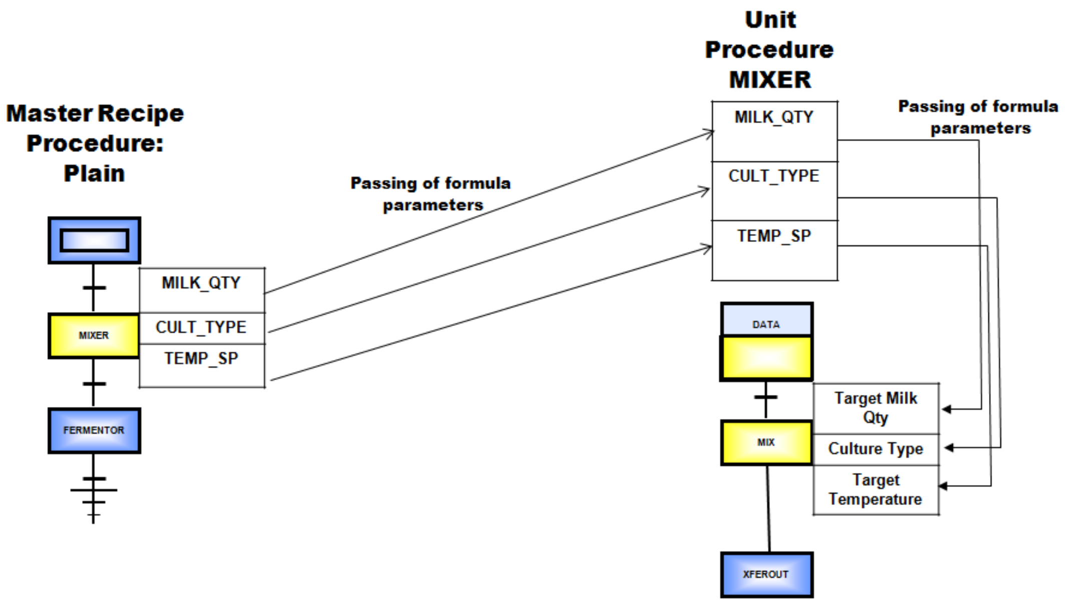


Figure 9: Data block in a *Recipe*/SCM

The following figures illustrate how Phase blocks and Data blocks are used for passing formula and report parameters.



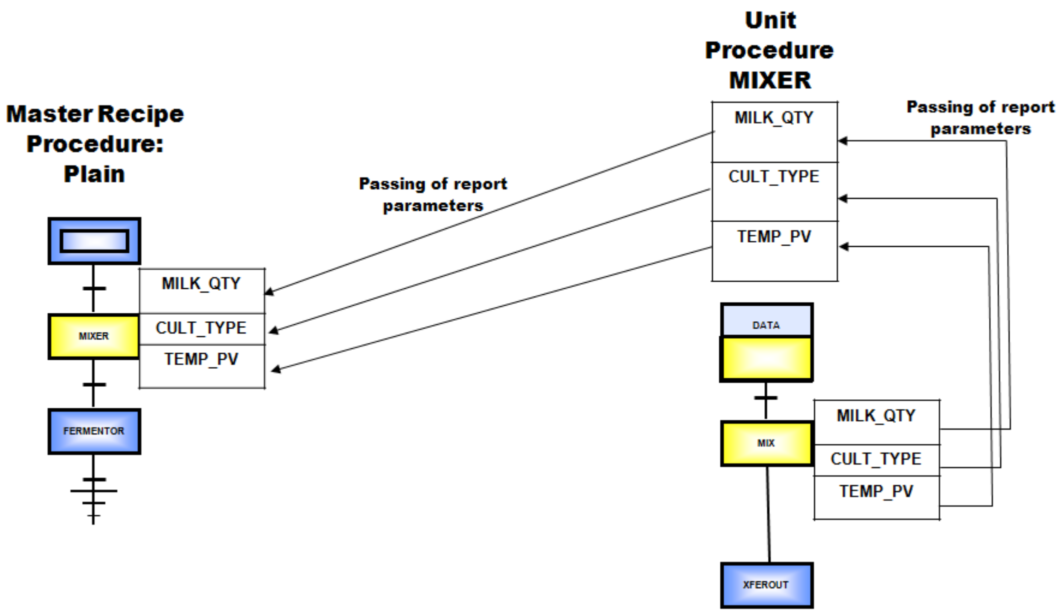


Figure 10: Phase blocks and Data blocks used for passing formula and report parameters

Note
Formula and report parameters support the following data types: BOOL, FLOAT64, INT32, STRING, and ENUM.

3.2.4 Instance-based recipes

Instance-based recipes are created to run against a specific Unit designated at build time. A Phase block contained by an RCM has a fixed relationship with the RCM or the SCM to be invoked.

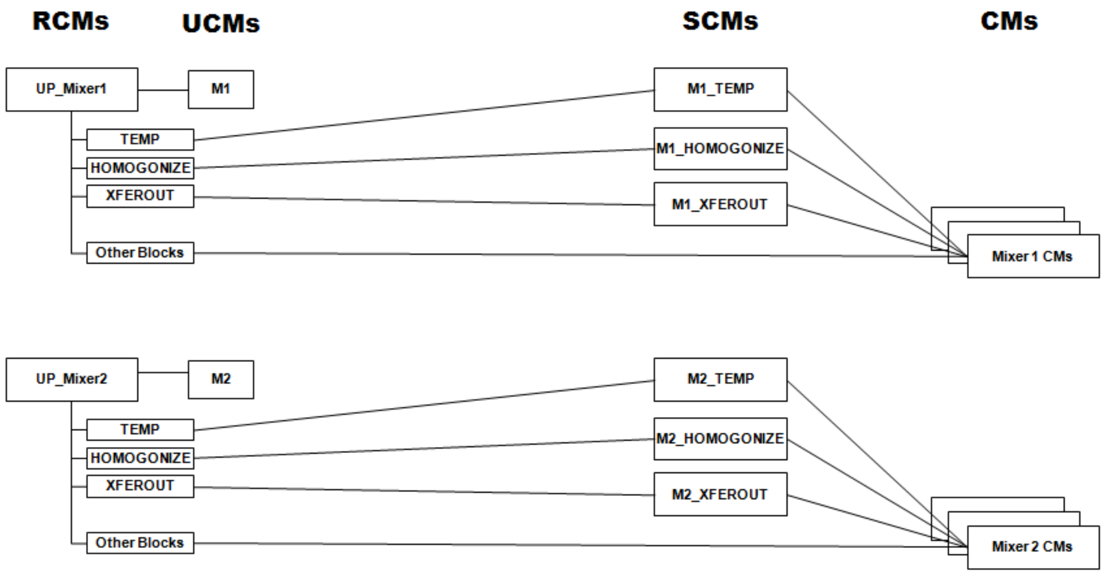


Figure 11: Example of an instance-based recipe

3.2.5 Class-based recipes

Class-based recipes function allows you to build a single recipe against a group of similar units, and instantiate multiple instances at run-time. These instances can be executed simultaneously against different units. The recipe is built once, tested once, and then run as many instances as needed (within the system limits). Without Class-based recipes function, you have to build a recipe for every single unit instance and maintain a large number of recipes. Each recipe instance must be tested individually, so the testing effort could be significant. Class-based recipes function helps reduce the cost on recipe engineering, maintenance, and testing.

Class-based Master Recipes and Class-based Control Recipes

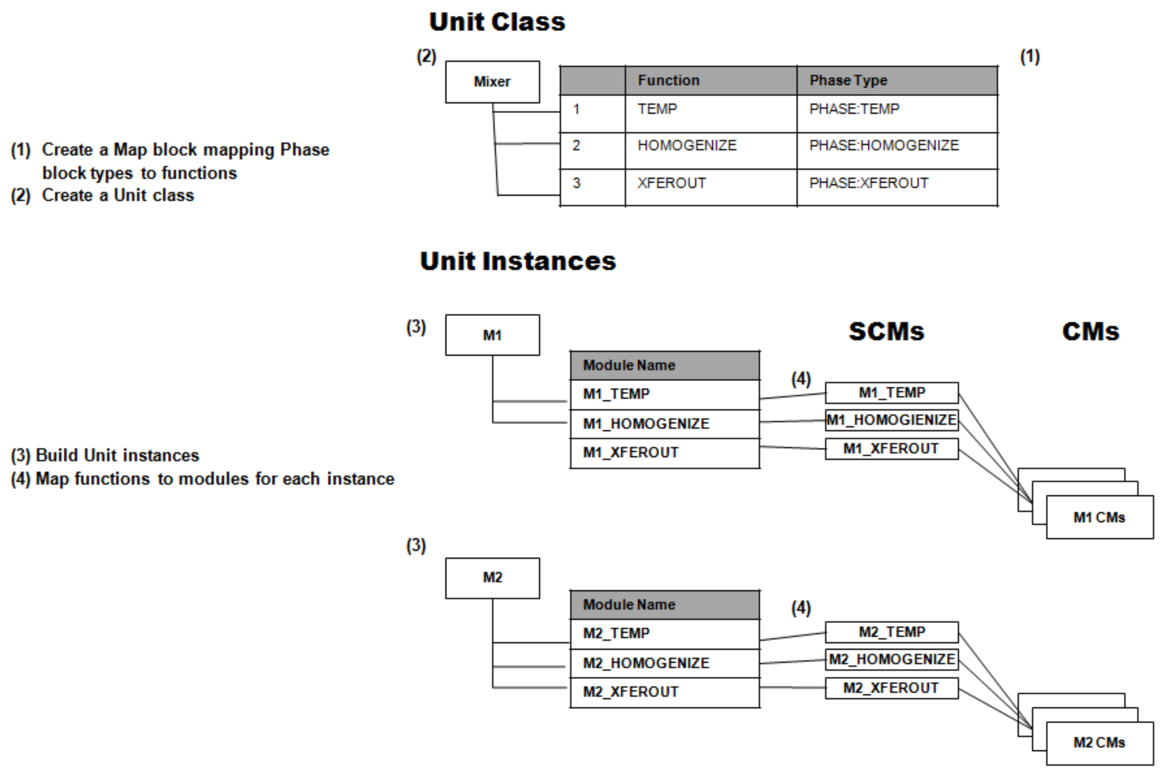
Class-based Master Recipes are built against Unit classes. They serve as "recipe templates."

Control Recipes are instantiated from Master Recipes and they are the executable version of the Master Recipes. A Control Recipe is associated with a specific Unit instance selected by the operator/program, and executes against that Unit instance at run-time.

Note

Although On-Process Migration (OPM) of controllers is supported in Experion, OPM is not supported for Class-based recipes when the Control Recipes are executing.

The following illustrations provide an overview of building EBM Class-based recipes.



	Function	Phase Type
1	TEMP	PHASE:TEMP
2	HOMOGENIZE	PHASE:HOMOGENIZE
3	XFEROUT	PHASE:XFEROUT

Module Name
M1_TEMP
M1_HOMOGENIZE
M1_XFEROUT

Module Name
M2_TEMP
M2_HOMOGENIZE
M2_XFEROUT

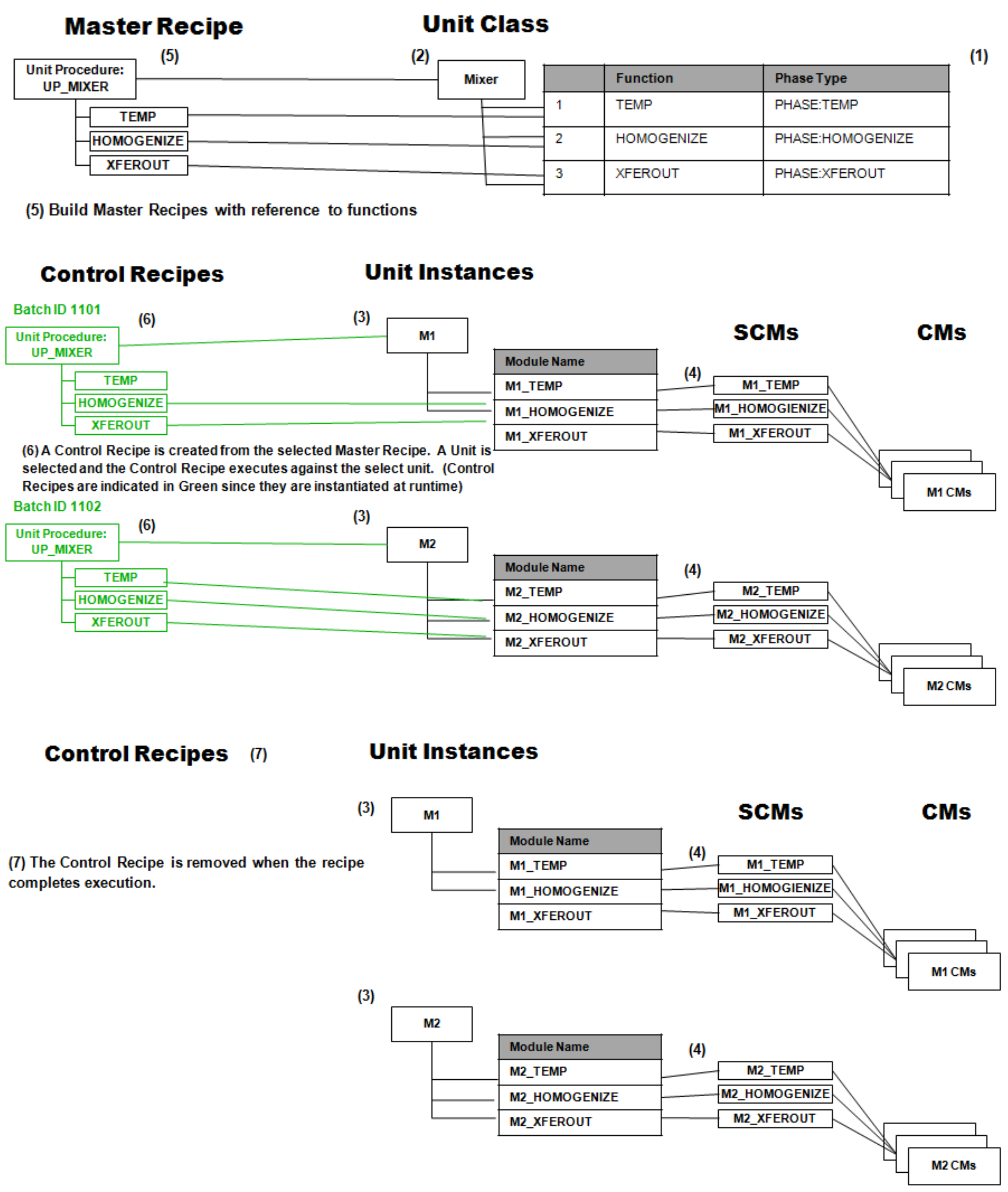

Master Recipe
Unit Class
(5)
(2)
(1)
Unit Procedure: UP_MIXER
Mixer
TEMP
HOMOGENIZE
XFEROUT
Function
Phase Type
1 TEMP PHASE:TEMP
2 HOMOGENIZE PHASE:HOMOGENIZE
3 XFEROUT PHASE:XFEROUT
(5) Build Master Recipes with reference to functions
Control Recipes
Unit Instances
Batch ID 1101
(6)
(3)
M1
Module Name
M1_TEMP
M1_HOMOGENIZE
M1_XFEROUT
SCMs
CMs
(4)
M1_HOMOGIENIZE
M1 CMs
(6) A Control Recipe is created from the selected Master Recipe. A Unit is selected and the Control Recipe executes against the select unit. (Control Recipes are indicated in Green since they are instantiated at runtime)
Batch ID 1102
M2
M2_TEMP
M2_HOMOGENIZE
M2_XFEROUT
M2 CMs
(7)
(7) The Control Recipe is removed when the recipe completes execution.

Figure 12: Example for building Class-based recipes

Note

Note that both Master Recipe and Control Recipe are controller-resident.

3.2.6 About unit acquisition and passing

EBM supports a functionally complete set of operations that can be used for configuring, acquiring, passing, and releasing units. The units needed for the entire batch can be selected at the top level recipe, prior to the start of the batch. The top level recipe passes the units to the lower layers. You have the option to configure the recipes to acquire all units needed for the entire batch up front by the top level recipe, or acquire the units at each layer when they are needed.

For more information about unit acquisition and unit passing, refer to the "Unit acquisition" chapter in the *Batch Implementation Guide*.

3.3 Overview of Procedural Operations

Procedural Operations (ProcOps) is an Experion-based solution to configure, implement, execute, analyze and maintain online operating procedures. It is used to automate procedures that previously were executed manually using paper documents. ProcOps provides services that help users justify, prioritize, implement, and maintain effective automated procedures. The following are the benefits of using ProcOps.

- Improve safety and reduce incidents due to problems in execution, such as procedures are not followed, procedures are lack of clarity
- Improve consistency of procedure execution
- Increase speed of procedure execution
- Improve collaboration between console and field operators
- Reduce operator work load when executing procedures so that the operator can focus on areas needing attention or intervention

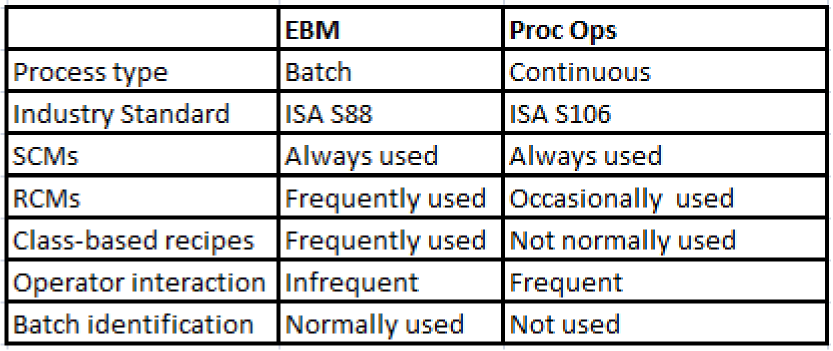
The function is normally carried out by SCMs with interactive instructions. RCMs may be used to coordinate execution of SCMs.

ProcOps function is normally used in continuous process to handle special operations such as plant start up and shut down.

Relationship between EBM and ProcOps

EBM uses Master Recipes, RCMs, and SCMs and applies them to batch processes. There are similar requirements for automated procedures in continuous processes which can also use RCMs and SCMs. There are ISA standards for the automation of batch and continuous processes: S88 and S106 respectively. Batch terminology is not used in continuous processes and so the term Procedural Operations or ProcOps is applied to the use of these features for continuous processes. There is no difference in the use of Experion between these two different applications.

The following table lists some of the differences between batch and ProcOps.



	EBM	**Proc Ops**
Process type	Batch	Continuous
Industry Standard	ISA S88	ISA S106
SCMs	Always used	Always used
RCMs	Frequently used	Occasionally used
Class-based recipes	Frequently used	Not normally used
Operator interaction	Infrequent	Frequent
Batch identification	Normally used	Not used

4 Overview of EBM Equipment model

Equipment model building consists of defining the following.

- Creating a Phase block type to define the following:
  - a "function" to be used by Unit class definition and recipes.
  - the formula and report parameter structure for the "function."
- Configuring the functions that need to be executed by a unit.
- Building Unit classes for organizing plant equipment into classes from which you can build Class-based Master Recipes and instantiate Unit instances.
- Instantiating Unit instances from Unit classes. At runtime, Control Recipes execute against the units.

The following figure illustrates an example equipment model.

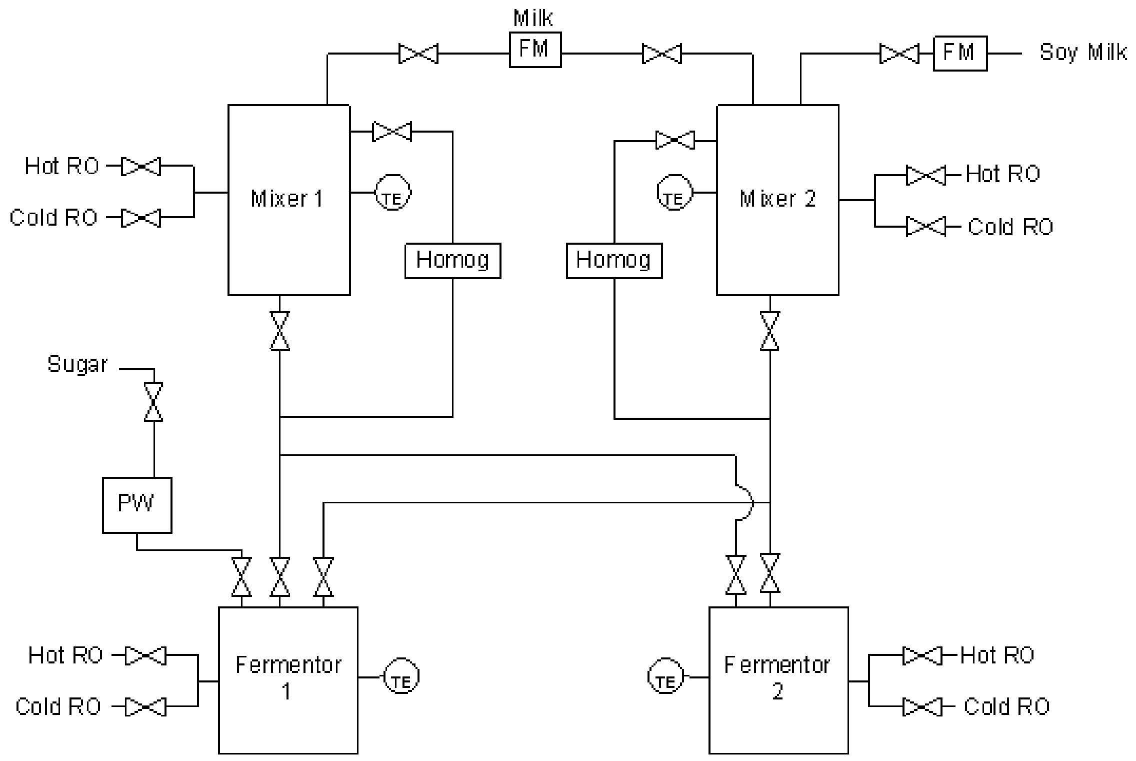


Figure 13: Example equipment model for Yogurt production

As illustrated in this example, there are two types of units: Mixer and Fermentor. Homogenize (Homog) is one of the functions that need to be executed by Mixers.

Related topics

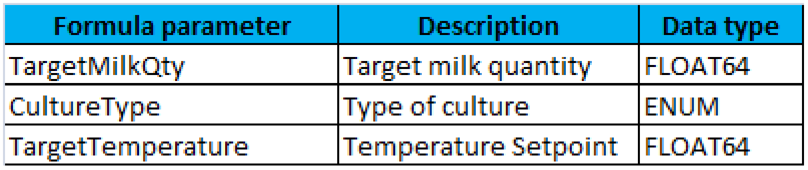
"About building Phase block types" on page 31

“About Unit classes” on page 32
“About Map block types” on page 33
“About Unit instances/UCMs” on page 34

4.1 About building Phase block types

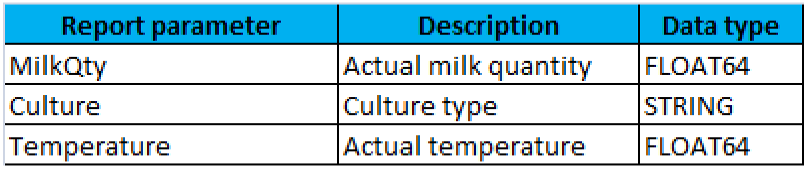
As a first step to build an equipment model, you need to build Phase block types that may be shared across Unit classes/Unit instances. As mentioned in one of the prior sections, Phase block types are used for defining formula and report parameters of functions contained by Unit classes/Units, and those functions define the "capabilities" of the Unit Class/Unit instances. In the yogurt production example, our Mixer Unit class can perform different functions. The functions for our example are two Material Additions, Temperature Control, Homogenize, and Transfer Out. Hence, the following Phase block types can be created for the functions: HOMOG (for homogenize function), MAT_ADD (for two material addition functions), TEMP_CTL (for temperature control function), XFR_OUT (for the transfer out function).

The following are the example formula parameters that can be created for the HOMOGENIZE Phase block type.



Formula parameter	Description	Data type
TargetMilkQty	Target milk quantity	FLOAT64
CultureType	Type of culture	ENUM
TargetTemperature	Temperature Setpoint	FLOAT64

The following are the example report parameters that can be created for the HOMOGENIZE Phase block type.



Report parameter	Description	Data type
MilkQty	Actual milk quantity	FLOAT64
Culture	Culture type	STRING
Temperature	Actual temperature	FLOAT64

Note

Formula and report parameters support the following data types: BOOL, FLOAT64, INT32, STRING, and ENUM.

For more information about building Phase block types, refer to the *Sequential Control User's Guide*.

4.2 About Unit classes

Unit classes are used for organizing plant equipment into classes from which you can build Class-based Master Recipes and instantiate Unit instances. Unit classes provide the necessary functions and references for building Master Recipes. If you refer to the example equipment model for yogurt production, Mixer 1 and Mixer 2 units can be grouped to form the Mixer Unit class. The units Fermentor 1 and Fermentor 2 can be grouped to form the Fermentor Unit class. The functions for our Mixer Unit class are two Material Additions, Temperature Control, Homogenize, and Transfer Out. The following figure illustrates the functions of the Mixer Unit class.

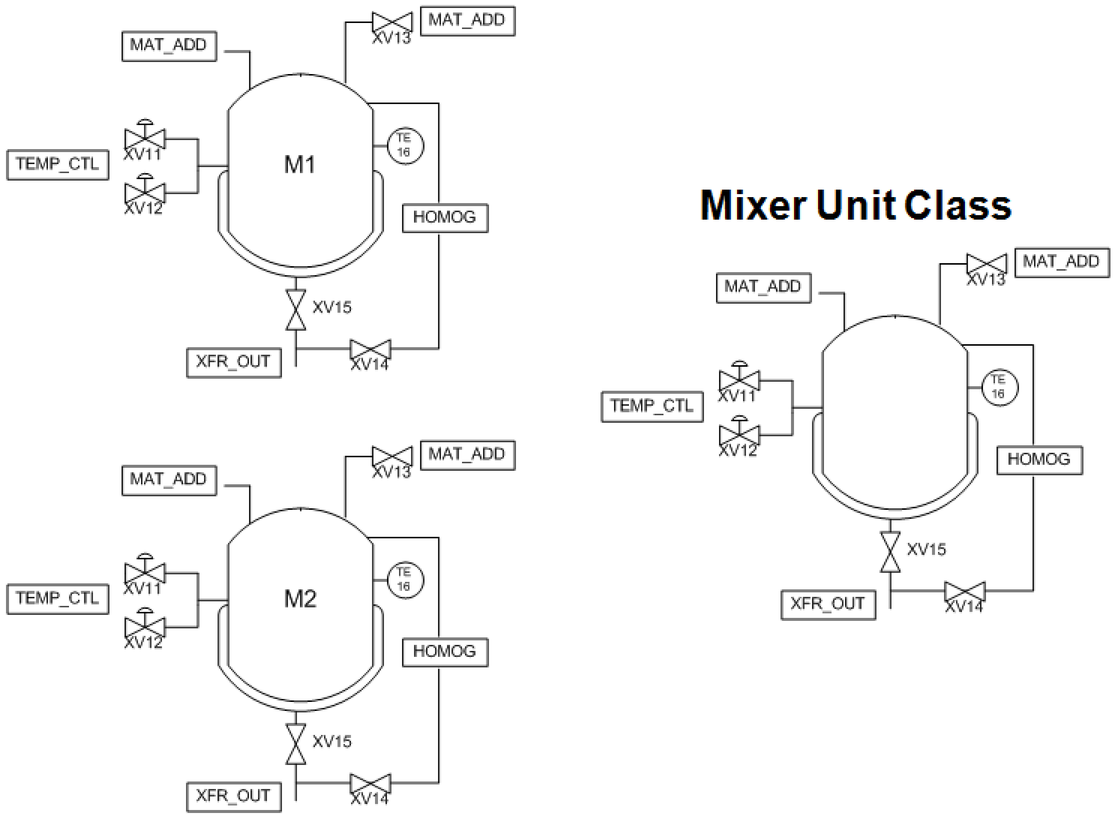


Figure 14: Example Unit class with functions

Note

If you have only one unit in your process cell, there is no need to create a Unit class. However, as a best practice and considering future expansions, it is recommended to create a Unit class even if you have one unit.

4.3 About Map block types

Map block types define functions and references that can be contained by Unit classes/Unit.

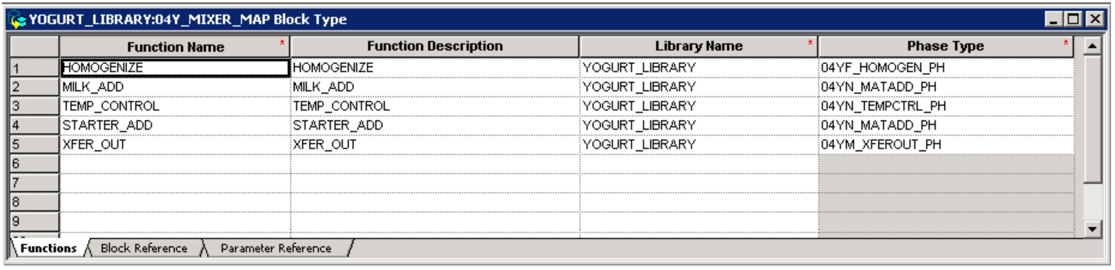
Map blocks contained by Unit classes have the "structural" definition of functions and references. Typically, the functions and references are not linked to any specific modules, blocks, or parameters. These functions and references are resolved to specific modules (RCMs/SCMs) and container blocks/basic blocks/parameters respectively when a Unit instance is instantiated from the Unit class. Also, these functions and references are used to configure a Master Recipe.

Note
Map blocks are contained in Unit classes/UCM templates.

Map block types support the following types of references.

- Function - Defines the functions that may be executed by the Unit class/Unit.
- Block Reference - Defines a block level reference associated with the Unit class/Unit. One reference provides access to all parameters of the block.
- Parameter Reference - Defines a parameter reference associated with the Unit class/Unit.

As illustrated, the PDE is used to create a Map block type (MIXER_MAP) and define functions and references for the block type. These functions and references are then a part of the Unit class when the Map block type is dropped into the Unit class.


YOGURT_LIBRARY:04Y_MIXER_MAP Block Type

	Function Name *	Function Description	Library Name *	Phase Type *
1	HOMOGENIZE	HOMOGENIZE	YOGURT_LIBRARY	04YF_HOMOGEN_PH
2	MILK_ADD	MILK_ADD	YOGURT_LIBRARY	04YN_MATADD_PH
3	TEMP_CONTROL	TEMP_CONTROL	YOGURT_LIBRARY	04YN_TEMPCTRL_PH
4	STARTER_ADD	STARTER_ADD	YOGURT_LIBRARY	04YN_MATADD_PH
5	XFER_OUT	XFER_OUT	YOGURT_LIBRARY	04YM_XFEROUT_PH
6				
7				
8				
9				

Figure 16: Example of a Map block type configuration

For more information about these references supported by Map block types and how to configure Map block types, refer to the *Batch Implementation Guide*.

4.4 About Unit instances/UCMs

Unit instances are strategies instantiated from a Unit class.

As mentioned earlier, functions and references defined in a Map block are resolved to specific modules (*Recipes*/SCMs) and container blocks/basic blocks/parameters respectively when a unit instance is instantiated from the unit class. The Control Recipes execute against Unit instances selected by an operator prior to the start of the Control Recipe execution. The following figure illustrates how the functions defined in a Map block are resolved by Unit instances.

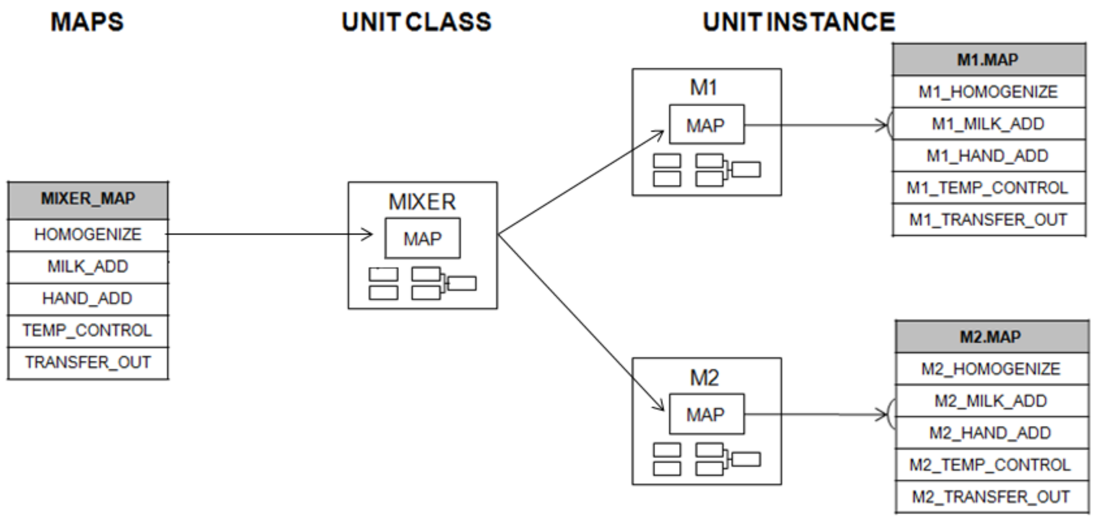


Figure 17: Example of Unit instantiation from Unit classes

Note

Unit Classes are equivalent to UCM templates, and Unit instances are equivalent to UCMs.

For more information about building Unit instances, refer to the *Batch Implementation Guide*.

5 Build-time support - Engineering tools

Control Builder and Recipe Builder are the engineering tools supported in EBM. Control Builder is used for building/configuring control strategies and equipment model. Recipe Builder is used for recipe building.

Recipe Builder provides a simplified building environment for recipe building. It is intended to be used by formulators who do not require an extensive tool like Control Builder for building recipes and who do not have access to Control Builder. However, for the formulators to gain access to Recipe Builder, system administrators must enable the access from Station. To use the Recipe Builder, control engineers must set the necessary access permissions from Control Builder. For more information, refer to the topics “About enabling Formulators to access Recipe Builder” on page 52 and “About setting permission levels for using Recipe Builder” on page 53.

Note that Recipe Builder supports only the tasks and objects that are required for recipe building. However, Recipe Builder uses the same locking mechanism as Control Builder. This ensures that objects cannot be edited simultaneously from both the tools.

Differences between Recipe Builder and Control Builder

The following are some of the major differences between Recipe Builder and Control Builder.

Recipe Builder	Control Builder
Following operations are not supported in Recipe Builder. • Export • Import • Bulk Edit • Bulk Build	All operations are supported.
Configuring and editing of Control Modules, UCMs, and Unit classes is not supported.	Configuring and editing of Control Modules, UCMs, and Unit classes is supported.
Only basic QVCS Manager functions are supported from Recipe Builder.	All QVCS Manager functions are supported from Control Builder.
Recipe Builder supports version control only for the recipe-related objects.	Control Builder supports version control for all objects.
Containment cannot be performed for UCMs and CMs from Recipe Builder.	Containment can be performed for UCMs and CMs from Control Builder.

6 Overview of Activities

Overview of Activities concept

Activity is a mechanism used to “frame” a series of actions that occurs in a plant within a defined beginning and end time period. An activity is capable of representing everything from a complex series of actions, (for example, a batch that makes products across multiple units), to a limited set of actions focused on a specific task, (for example, adding material to a mixer). Activities are associated with *Recipes*/SCMs, and may be built from a series of Parent and Child Activities dependent on the level of modularity and complexity of the actions being framed. An Activity is also meant to represent the entire lifecycle of a Batch (Pre-Execution, Execution, and Post-Execution), or a series of actions (such as Batch created, Batch started, Batch executed, Batch completed, and Batch removed). An Activity is equivalent to a Batch in batch applications, or a Procedure in Procedural Operations.

Overview of Activities function

The purpose of Activity function is to provide a common infrastructure to represent “Activities” (as defined in Overview of Activity Concept section) based on entities from different applications, such as Batch Automation, Procedural Operation, Blending and Movement Automation, or Honeywell Field Advisor, in theExperion system. It provides a means to support common application integration. The current product focuses on the integration with Batch applications and Procedural Operations. Activity function provides a mechanism and interfaces to allow operators, internal and external applications to create, monitor, command, and remove Batches/Procedures/Activities from the Experionsystem. It also provides a consolidated view of Batches/ Procedures/Activities that are part of operator's responsibilities.

Related topics

“Activities-related Operational UIs” on page 38

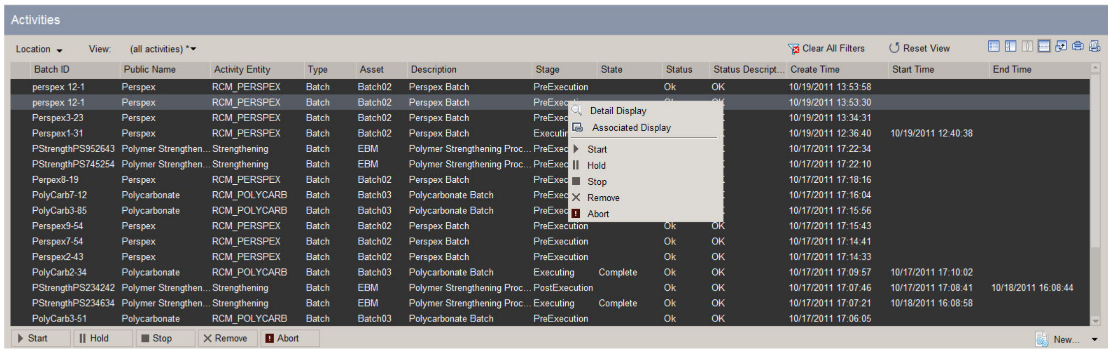
6.1 Activities-related Operational UIs

Summary displays

The summary displays provides a summary view of the top level Activities (or Batches, Procedures) existing in a system. It displays key information of Activities, such as State, Status, and Stage, supports creating and commanding Activities. It can also be used for viewing and updating the header, formula and report parameters of Activities, and navigating to their detail displays and associated displays. The summary displays are intended to be the main interface for user interactions with individual Activities (or Batches, Procedures). The summary displays are delivered in three different forms:

- Batch Summary Display which provides a summary view of top level Batches in Pre-Execution, Execution, and Post-Execution stage, and is typically used in Batch applications.
- Procedure Summary Display which provides a summary view of top level Procedures in Pre-Execution, Execution, and Post-Execution stage and is typically used in Procedural Operations.
- Activity Summary Display which provides summary view of all top level Activities (Batches and Procedures) in Pre-Execution, Execution, and Post-Execution stage.

These displays are supported as standard Experion displays running on Experion Station. They are also supported on Experion Console Station when the server is available. Batch, Procedure, and Activity Summary Displays have a similar look and operation to other summary displays in the system, such as Alarm, Event, Alert, and Message Summary Display. They support standard display functions, such as sorting and filtering. The following figure illustrates the Activity Summary Display.

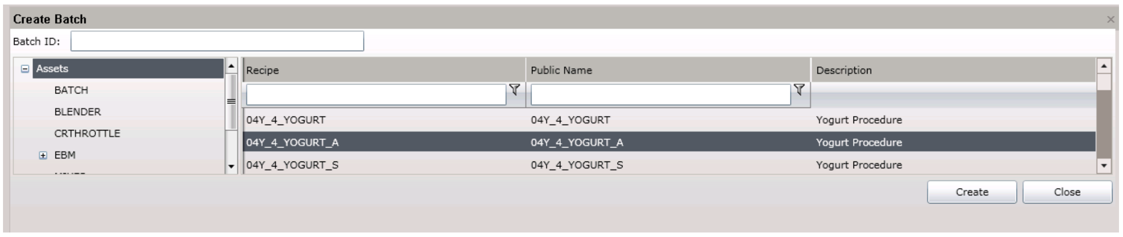


Batch ID	Public Name	Activity Entity	Type	Asset	Description	Stage	State	Status	Status Descript...	Create Time	Start Time	End Time
perspex 12-1	Perspex	RCM_PERSPEX	Batch	Batch02	Perspex Batch	PreExecution		Ok	OK	10/19/2011 13:53:58		
perspex 12-1	Perspex	RCM_PERSPEX	Batch	Batch02	Perspex Batch	PreExec				10/19/2011 13:53:30		
Perspex3-23	Perspex	RCM_PERSPEX	Batch	Batch02	Perspex Batch	PreExec				10/19/2011 13:34:31		
Perspex1-31	Perspex	RCM_PERSPEX	Batch	Batch02	Perspex Batch	Executin				10/19/2011 12:36:40	10/19/2011 12:40:38	
PStrengthPS952643	Polymer Strengthen...	Strengthening	Batch	EBM	Polymer Strengthening Proc...	PreExec				10/17/2011 17:22:34		
PStrengthPS745254	Polymer Strengthen...	Strengthening	Batch	EBM	Polymer Strengthening Proc...	PreExec				10/17/2011 17:22:10		
Perpex8-19	Perspex	RCM_PERSPEX	Batch	Batch02	Perspex Batch	PreExec				10/17/2011 17:18:16		
PolyCarb7-12	Polycarbonate	RCM_POLYCARB	Batch	Batch03	Polycarbonate Batch	PreExec				10/17/2011 17:16:04		
PolyCarb3-85	Polycarbonate	RCM_POLYCARB	Batch	Batch03	Polycarbonate Batch	PreExec				10/17/2011 17:15:56		
Perspex9-54	Perspex	RCM_PERSPEX	Batch	Batch02	Perspex Batch	PreExecution		Ok	OK	10/17/2011 17:15:43		
Perspex7-54	Perspex	RCM_PERSPEX	Batch	Batch02	Perspex Batch	PreExecution		Ok	OK	10/17/2011 17:14:41		
Perspex2-43	Perspex	RCM_PERSPEX	Batch	Batch02	Perspex Batch	PreExecution		Ok	OK	10/17/2011 17:14:33		
PolyCarb2-34	Polycarbonate	RCM_POLYCARB	Batch	Batch03	Polycarbonate Batch	Executing	Complete	Ok	OK	10/17/2011 17:09:57	10/17/2011 17:10:02	
PStrengthPS234242	Polymer Strengthen...	Strengthening	Batch	EBM	Polymer Strengthening Proc...	PostExecution		Ok	OK	10/17/2011 17:07:46	10/17/2011 17:08:41	10/18/2011 16:08:44
PStrengthPS234634	Polymer Strengthen...	Strengthening	Batch	EBM	Polymer Strengthening Proc...	Executing	Complete	Ok	OK	10/17/2011 17:07:21	10/18/2011 16:08:58	
PolyCarb3-51	Polycarbonate	RCM_POLYCARB	Batch	Batch03	Polycarbonate Batch	PreExecution		Ok	OK	10/17/2011 17:06:05		

Figure 19: Activity Summary Display

Creation UI

The Creation UI, when invoked, provides a list of *Recipes*/SCMs for an operator to select and create Batches/ Procedures/Activities from the selected *Recipes*/SCMs. The Creation UI can be invoked from the Summary Display. It can also be invoked from the Custom Display. The following figure illustrates the Activity creation UI.

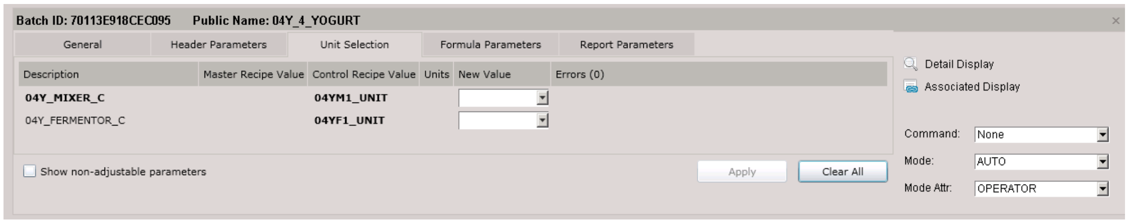


Recipe	Public Name	Description
04Y_4_YOGURT	04Y_4_YOGURT	Yogurt Procedure
04Y_4_YOGURT_A	04Y_4_YOGURT_A	Yogurt Procedure
04Y_4_YOGURT_S	04Y_4_YOGURT_S	Yogurt Procedure

Figure 20: Activity creation UI

Data UI

This UI displays formula/report parameters, unit selection, and header information of a selected Batch/Procedure/Activity. An operator can use this UI to set up batch size, select unit, change the adjustable formula parameter values, and enter report data. The Data UI can be invoked from the Summary Display, and custom displays. The following figure illustrates the Activity Data UI.



Batch ID: 70113E918CEC095 Public Name: 04Y_4_YOGURT

General | Header Parameters | Unit Selection | Formula Parameters | Report Parameters

Description	Master Recipe Value	Control Recipe Value	Units	New Value	Errors (0)
04Y_MIXER_C		04YM1_UNIT			
04Y_FERMENTOR_C		04YF1_UNIT			

Show non-adjustable parameters

Apply | Clear All

Detail Display

Associated Display

Command: None

Mode: AUTO

Mode Attr: OPERATOR

Figure 21: Activity Data UI

Activity/Batch/Procedure Detail Display

Similar to the RCM/SCM Detail Display, this display provides detailed information about a Batch/Procedure/Activity. From this display, an operator can view formula/report parameters and sequence charts, change adjustable formula parameter values, and enter report data. The operator can also view the runtime data when the Batch/Procedure/Activity is executing and also navigate to a child recipe details. The Detail Display can be invoked from the Summary Display.

Custom Display

The **Activity Table** and the **Create Activity** pushbutton can be configured specifically for batch only, procedural operations only, or a mixture of both.

There are several controls available on the custom display to creating, executing, and monitoring activities. For more details, refer to the *Experion Operator's Guide*. The following figure illustrates a sample custom display.

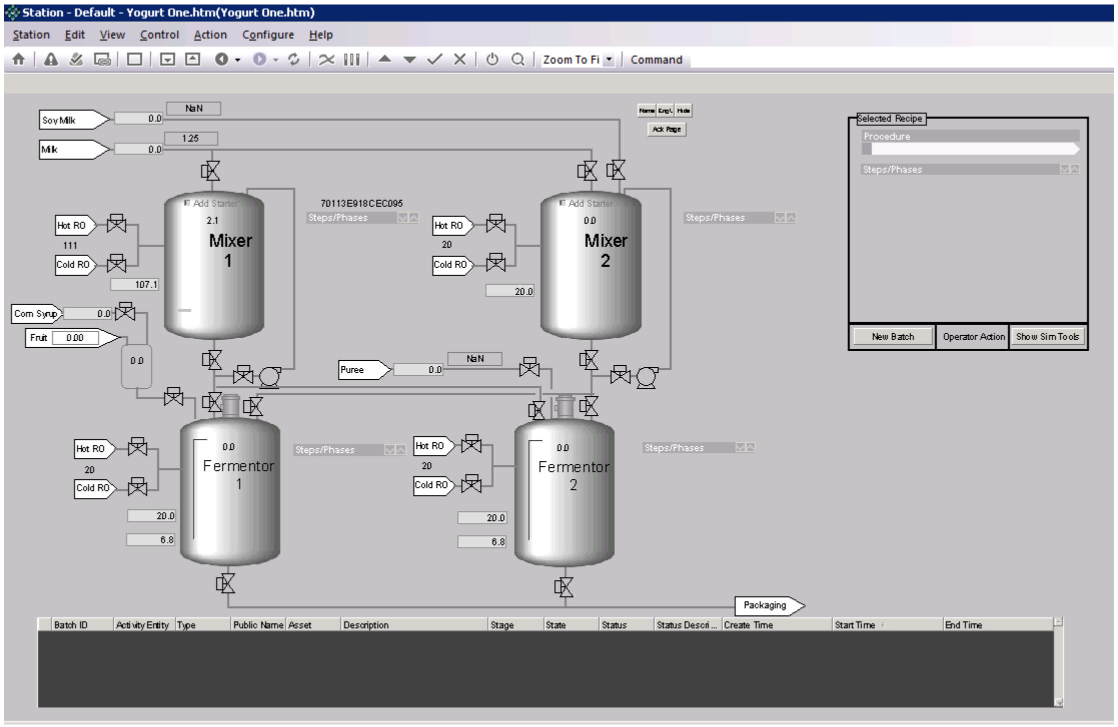


Figure 22: Sample custom display

7 Batch Application Services

Batch Application Services provide a set of Application Programming Interfaces (APIs) that allow an application (for example, MES) to create, control, and interact with batches/procedures/activities across Experion systems, including DSA-connected systems. The APIs provide programmatic access to create, command, monitor, update, and remove top-level batches/procedures/activities. Access is similar in scope to that of an operator using Station.

For more information about Batch Application Services, refer to the *Application Development Guide*.

8 Monitoring, diagnostics, and post-execution support

About monitoring and diagnostics support

A set of statistic/diagnostic parameters are provided to assist you in monitoring and troubleshooting your batch/procedure execution. For more details, refer to the *Batch Implementation Guide*.

Post-execution support

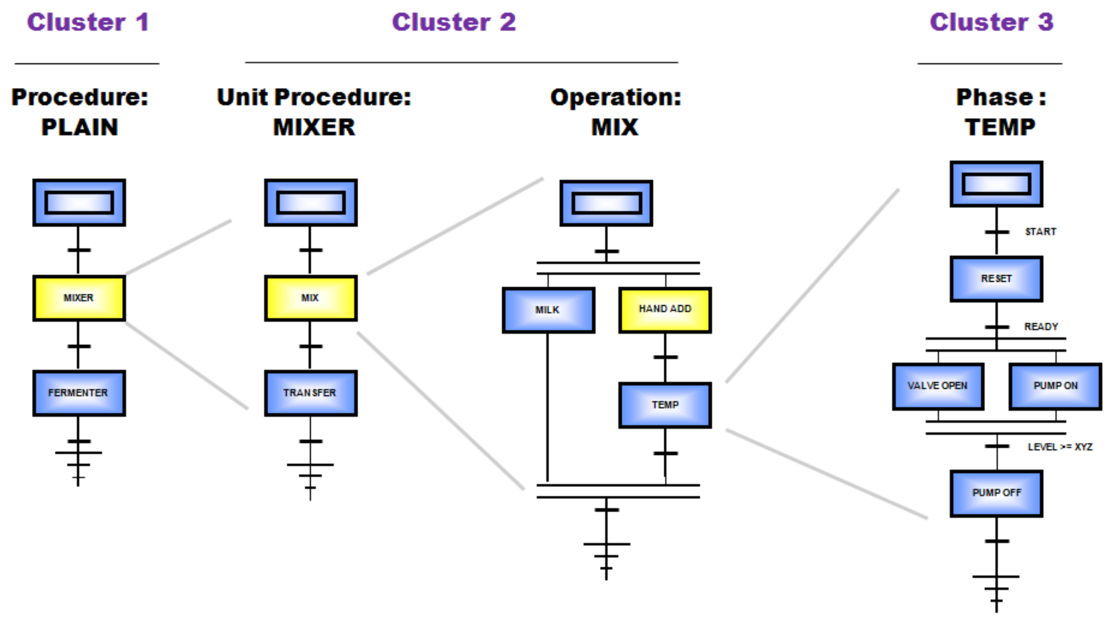
Batch/procedure execution history is captured by a set of raw batch events. These events may be used by programs (that is, Procedure Analyst) to generate reports and perform post-execution analysis. For more information about batch events, refer to the *Sequential Control User's Guide*.

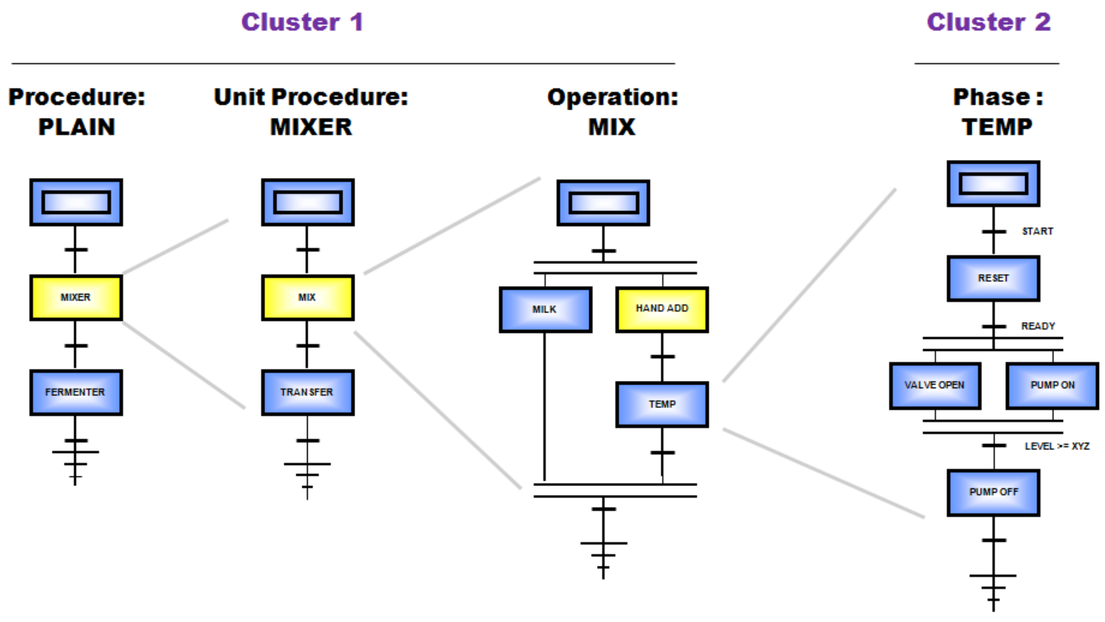
Procedure Analyst is a comprehensive batch/procedure reporting and analysis tool. It collects and processes batch/procedure events and continuous information pertaining to batch/procedure execution. After processing, the Procedure Analyst Desktop Tools allow users to query, report and trend (in conjunction with Uniformance Process Studio) batch/procedure information. Procedure Analyst integrates batch/procedure event data with continuous and Experion event data. Refer to the Procedure Analyst Specifications for more details.

9 Overview of intercluster peer-to-peer communication

Intercluster peer-to-peer function enables communication between batch layers distributed in two or more Experion clusters. This function allows a parent recipe in one cluster to control its child recipes in another cluster. The clusters participating in intercluster peer-to-peer communication may be in the same or different FTE communities.

The following figures illustrate how intercluster peer-to-peer communication can be configured between recipe layers:





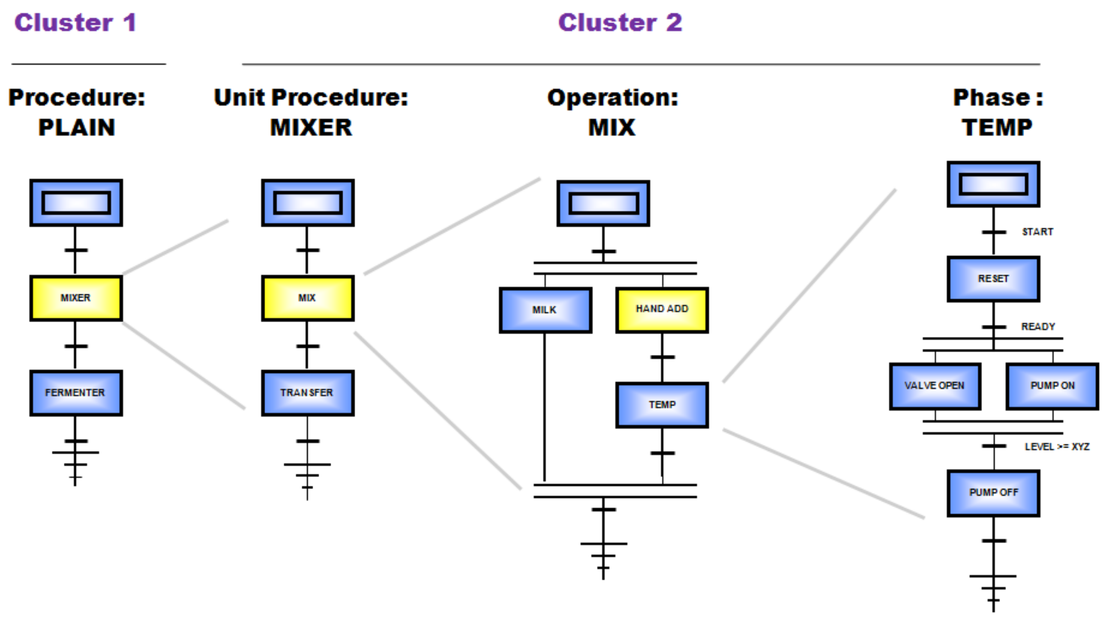


Figure 23: Intercluster peer-to-peer examples

For more information on configuring intercluster peer-to-peer communication, refer to the *Batch Implementation Guide*.

10 General planning guidelines

Related topics

"Size of a project" on page 48
"Licensing considerations" on page 49
"Distribution across multiple Control Execution Environments" on page 50
"Scope of Responsibility considerations" on page 51
"About enabling Formulators to access Recipe Builder" on page 52
"About setting permission levels for using Recipe Builder" on page 53

10.1 Size of a project

Before you begin batch implementation, it is important to determine the size of your project, understand the EBM licensing model and decide what licenses you require for your project. To determine the size of your project, you need to assess the following:

1. Determine the hardware required: Number of servers, stations, controllers and the number of process points (based on I/O). You may refer to the *Site Planning Guide* and the *Control Hardware Planning Guide*.
2. Determine the number of S88 Units. One Unit instance is required for each Unit and each Unit instance consumes one process license point. Also, each SCM, RCM, and Master Recipe consume one license point.
3. Determine the number of SCMs required. You may refer to the *Sequential Control User's Guide*.
4. Determine the number of recipes required and determine how many Units are involved in each recipe.
5. Check if the number of process points in step 1 is sufficient for any spare requirement. If it is not sufficient, you may need to buy additional licenses.
6. Determine how many recipes will be running concurrently. From this count, determine how many RCMs will be executing concurrently.
7. Review your estimate with your Honeywell representative.

Note

For more information on determining the size of your project, work with your Honeywell representative.

10.2 Licensing considerations

In EBM, two sets of batch license models are available (with and without Class-based recipes).

EBM is licensed per server. License instances can be distributed between any CEE assigned to the server. Multiple servers are supported; but, limited by the license instances on each server. The license count is evaluated by monitoring the sum of the S88 procedural elements: Procedure, Unit Procedure and Operation that are concurrently executing. This number is only evaluated when a batch is executing (that is, for batches that are created and for batches that are not in pre execution or post execution). A system alarm is raised if the license is exceeded. However, you are never prevented from creating or starting a batch because a license limit has been exceeded.

A procedural element such as a Master Recipe or an RCM at any level, when loaded to a CEE, consumes a single process point license. A batch created from a loaded Master Recipe or an RCM appears in the summary displays but consumes no additional point licenses. EBM instances are used only when a procedural element is executing.

Recipe Builder license

Recipe Builder provides a subset of functionality available in Control Builder for building recipes. For licensing purposes, Recipe Builder is treated as an instance of Control Builder. There is no specific license for Recipe Builder.

Note

For more information about EBM licensing, contact your Honeywell representative.

10.3 Distribution across multiple Control Execution Environments

The structure of EBM Recipe and equipment model provides you the flexibility of utilizing the centralized model. All recipe layers and unit definitions are placed in one CEE, or the Distributed Model, where recipe layers and unit definitions are distributed among CEEs within the entire Experion system. But in reality, purely centralized model and purely distributed model may not be practical due to the limitation of the core system capacity (controller memory and CPU, peer-to-peer communication bandwidth). The memory and CPU limitation of a controller may prevent you from running all recipe layers and units in one controller. On the other hand, the limitation of peer-to-peer communication bandwidth of the system may prevent you from placing every recipe layer in different controllers. To effectively manage the usage of system resources, you need to balance them when designing the batch systems, and special features, such as dynamic fetch with Enable/Disable Peer Requests function, whenever possible must be used.

Platforms

The execution of units and each of the recipe layers (Procedure, Unit Procedure, Operation, and Phase/SCMs) can be hosted on C300, C200E, ACE, and associated simulation platforms independently. In addition, the execution of Phase layer (SCMs) can also be hosted on C200 platform with the restriction that the host C200 controller must be in the same cluster with the CEE, where the parent operation layer resides. Recipes can be in the same or different controllers as their units. It is possible to assign units and recipe layers to required platforms based on specific performance and availability requirements. These units and recipes can always be reassigned to different controllers in the future to adjust memory usage, performance, and availability needs.

For the best performance, it is recommended that *Recipes* are assigned to the same controller as the UCMs against which they execute.

Attention

- EBM does not have a warm restart capability in the current release for recovering Recipe/Procedures from their point of execution upon recovery. You must carefully analyze which recipes, procedures, units are critical to run within a redundant controller and which are acceptable to execute within a non-redundant platform. This limitation also applies to allocation management of resources.
- Control Recipes and their Master Recipes are tightly coupled and must be in the same CEE.
- Unit passing across clusters is not supported. This factor must be considered when planning the distribution of batch layers across clusters. The general suggestion is to maintain Unit Procedures and their associated Operations in the same cluster.

10.4 Scope of Responsibility considerations

Scope of Responsibility of shared units

During the planning and design phase, the engineer should consider the Scope of Responsibility (SoR) for shared units, and make sure they are under the SoR of every operator who uses it. The following figure illustrates an example of a shared unit. Operator A is responsible for Unit1 and Operator B is responsible for Unit2. Operator A runs recipe group 1, which acquires a common operation (UnitOpComn) and a common phase (SCMComn). Both UnitOpComn and SCMComn run against Unit3. Operator B runs recipe group 2 which also acquires a common operation (UnitOpComn) and a common phase (SCMComn). Hence, Unit3 should be in the SoR of both Operator A and Operator B.

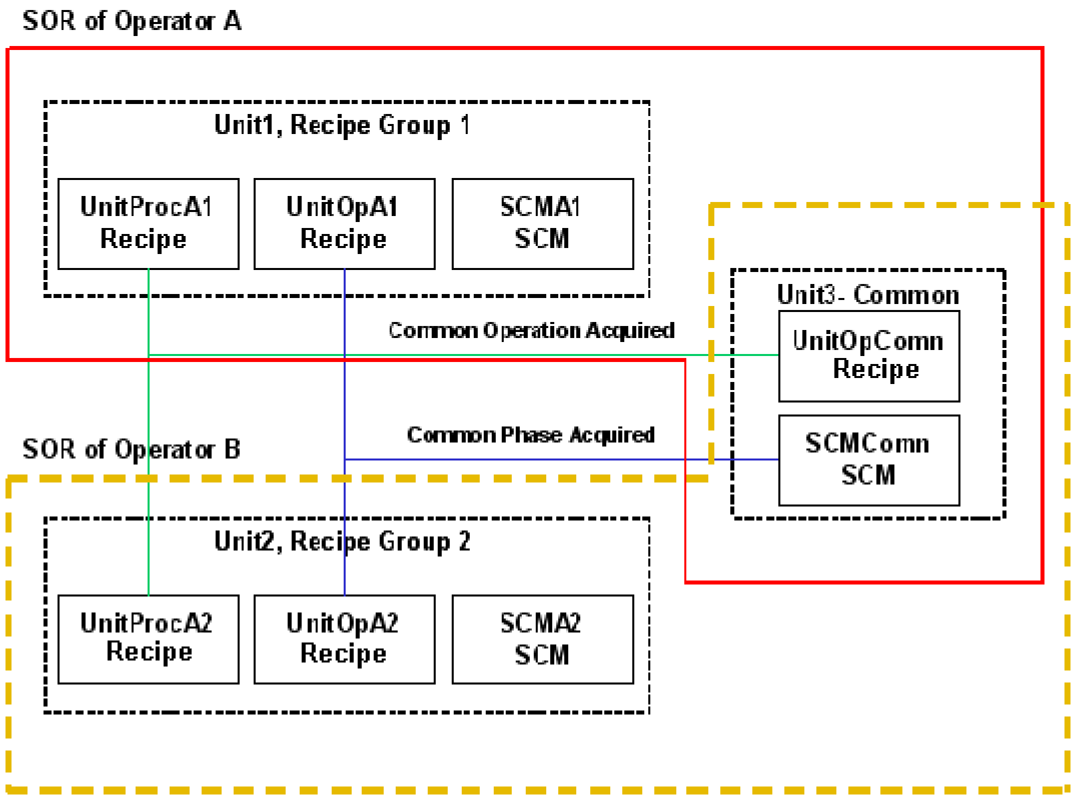


Figure 24: Scope of Responsibility considerations

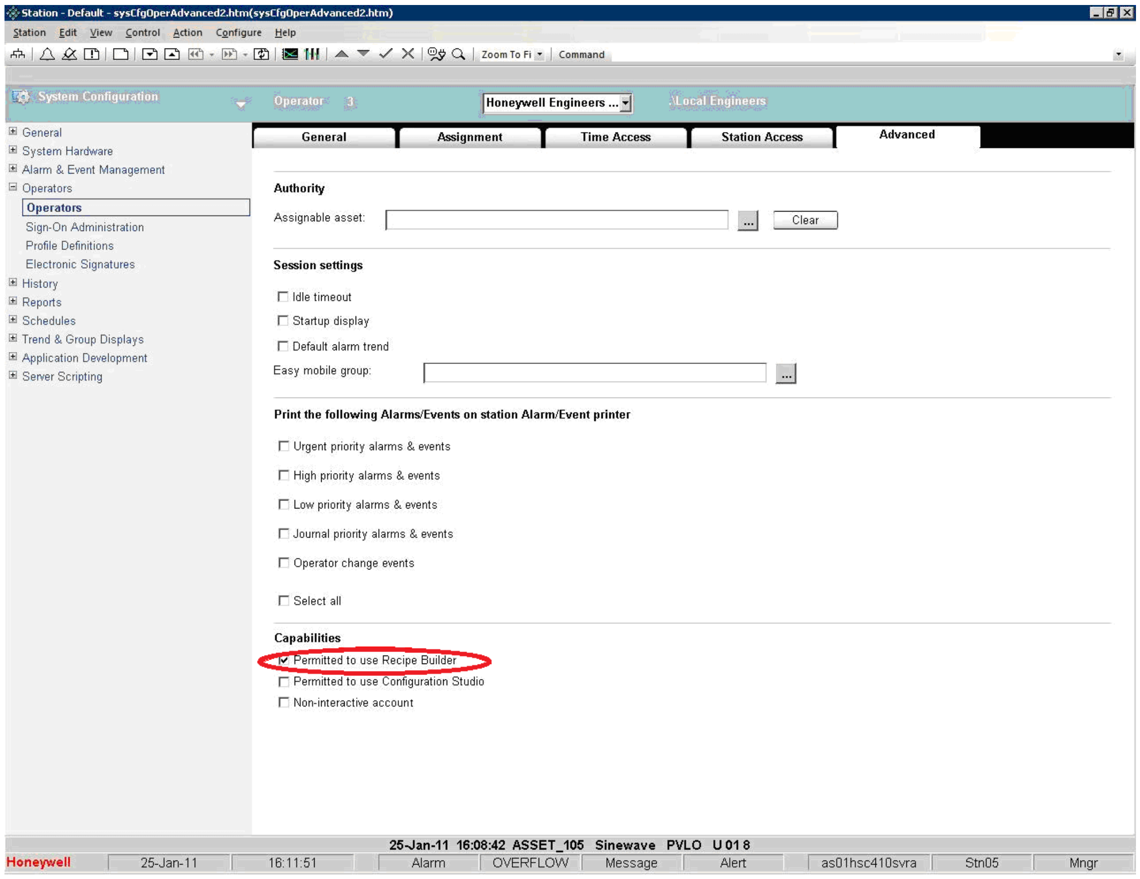
SoR planning for batches/procedures/activities

A batch/procedure/activity inherits the parent asset from its *Recipe*/SCM when created. For a batch/activity created from a Class-based Master Recipe, the asset remains the same as the asset of its Master Recipe prior to the primary unit selection. After selecting the primary unit, the asset of the batch/activity changes to the asset of the selected primary unit. The batch/activity is only under the SoR of the operator, if both the Master Recipe and the primary unit are under the SoR of the operator. For a batch/procedure/activity created from an RCM or SCM, the asset of the batch/procedure/activity is always the same as the asset of the RCM/SCM. If the RCM/SCM is under the SoR of the operator, the batch/procedure/activity created from the RCM/SCM is also under the SoR of the operator.

10.5 About enabling Formulators to access Recipe Builder

A system administrator can provide Formulators access to the Recipe Builder.

To enable access, navigate to the **Capabilities** page on the **Advanced** tab in the **Operator Configuration** display and select the option: **Permitted to use Recipe Builder**. The following figure illustrates the **Advanced** tab of the **Operator Configuration** display.



Authority

Assignable asset:

Session settings

- [ ] Idle timeout
- [ ] Startup display
- [ ] Default alarm trend

Easy mobile group:

Print the following Alarms/Events on station Alarm/Event printer

- [ ] Urgent priority alarms & events
- [ ] High priority alarms & events
- [ ] Low priority alarms & events
- [ ] Journal priority alarms & events
- [ ] Operator change events
- [ ] Select all

Capabilities

- [x] Permitted to use Recipe Builder
- [ ] Permitted to use Configuration Studio
- [ ] Non-interactive account

25-Jan-11 16:08:42 ASSET_105 Sinewave PVLO U 01 8

Honeywell | 25-Jan-11 | 16:11:51 | Alarm | OVERFLOW | Message | Alert | as01hsc410svra | Stn05 | Mngr

Figure 25: Operator Configuration display

Note

By default, users belonging to the Product Administrators, Local Engineers, and MNGR groups are provided access to Recipe Builder and Configuration Studio. However, users belonging to other groups such as Local Supervisors, Local Operators, Local Ack View Only Users, and Local View Only Users are not provided access.

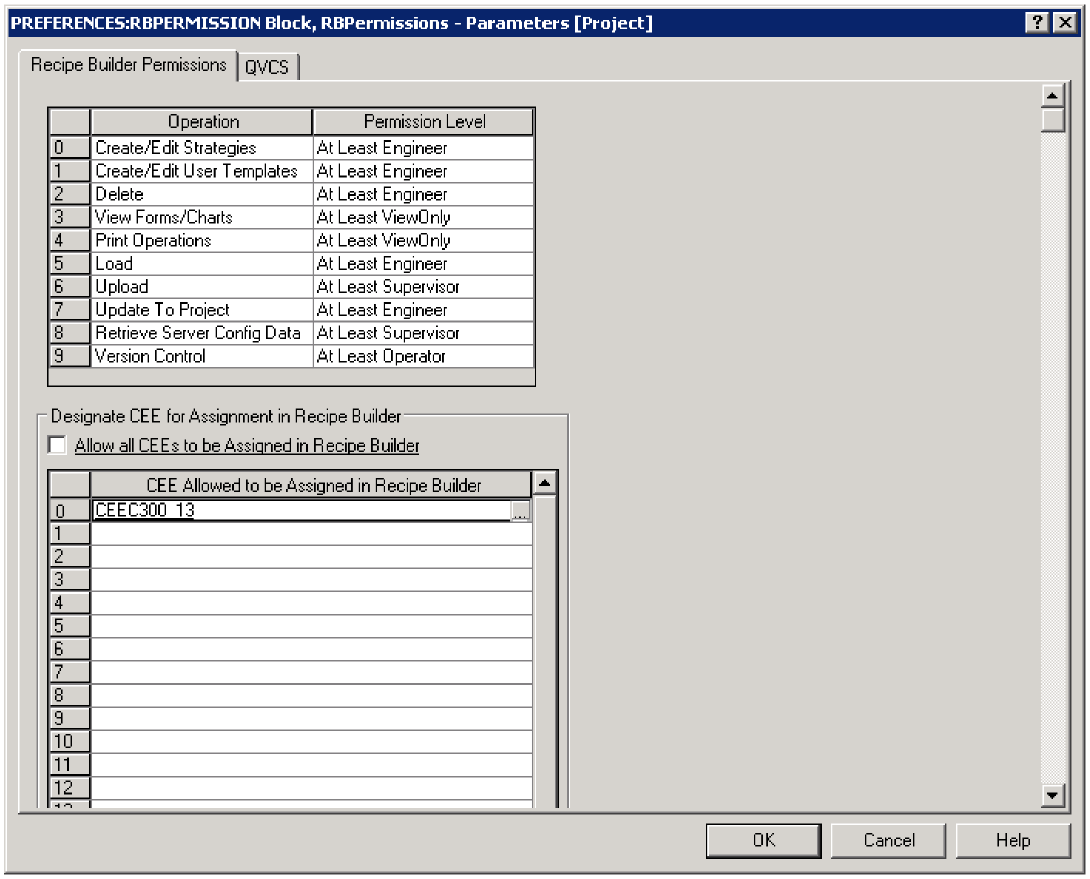
10.6 About setting permission levels for using Recipe Builder

Before using Recipe Builder for building recipes, the following needs to be configured from Control Builder by a control engineer.

- Set the minimum permission levels for performing each of the Recipe Builder operations
- Designate which CEEs can be used in Recipe Builder for assigning recipes

This configuration can be performed from the **Tools**->**Recipe Builder Operations Permissions** on the Control Builder. This option is similar to the **Tools**->**Operations Permissions** menu item supported by Control Builder.

The following figure illustrates the **Recipe Builder Permissions** dialog box.


PREFERENCES:RBPERMISSION Block, RBPermissions - Parameters [Project]
Recipe Builder Permissions
QVCS
Designate CEE for Assignment in Recipe Builder
Allow all CEEs to be Assigned in Recipe Builder
OK
Cancel
Help

	Operation	Permission Level
0	Create/Edit Strategies	At Least Engineer
1	Create/Edit User Templates	At Least Engineer
2	Delete	At Least Engineer
3	View Forms/Charts	At Least ViewOnly
4	Print Operations	At Least ViewOnly
5	Load	At Least Engineer
6	Upload	At Least Supervisor
7	Update To Project	At Least Engineer
8	Retrieve Server Config Data	At Least Supervisor
9	Version Control	At Least Operator

	CEE Allowed to be Assigned in Recipe Builder
0	CEEC300_13
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Figure 26: Recipe Builder permissions

- The **Allow all CEEs to be Assigned in Recipe Builder** check box enables all CEEs to be selected from the **Execution Environment Assignment** dialog box when assigning a *Recipe* to a CEE in Recipe Builder. This option is selected by default; which means, all CEEs are designated for assignment in Recipe Builder.
- The **CEE Allowed to be Assigned in Recipe Builder** list enables you to specify the CEEs to be selected from the **Execution Environment Assignment** dialog box when assigning a *Recipe* to a CEE in Recipe Builder. By default, this list is empty. You can use the Point Picker to select each CEE. Note that only CEEs are presented in the Point Picker list.
  - If the **Allow all CEEs to be Assigned in Recipe Builder** check box is selected, you cannot enter the CEEs in this list.
  - If after entering the CEEs in the list, you select the **Allow all CEEs to be Assigned in Recipe Builder** check box, the list of CEEs will be maintained, but the list will not have an affect on which CEEs are allowed to be assigned.

Attention

If you do not change the default values of these two options, you cannot assign an RCM or a Master Recipe to the CEE from Recipe Builder.

11 Guidelines/considerations for building equipment model

Related topics

"Formula or Report data characteristics" on page 56
"Considerations for Phase block types" on page 57
"Considerations for enumeration sets" on page 58
"Differences between a UCM and a Unit class" on page 59
"Usage of Map blocks" on page 60
"Considerations for modifying an equipment model" on page 62

11.1 Formula or Report data characteristics

The Formula/Report data structure provides the ability to define the items, together with the sequence/procedure logic, for how to make a product. These data items are often broken up into the following different areas.

- Formula Data structure specific to the product being produced, and can be consumed at all potential levels of Layered Recipes. This is often defined and managed at the parent/top level, and key parameters are sent down to child components as needed, and key results are captured and pulled up to parent components.
- Phase function block structure specific to the function being performed. It defines a clear contract of data between the requestor (Parent RCM/Control Recipe) and the consumer (child RCM/SCM/Control Recipe). This contract is independent of the details on how the child component actually completes the task. For example, a HEAT function may provide a target temperature and require a resulting temperature reach and time to reach, but it does not define how the action is performed. In this case, one Equipment Unit may use steam for heating and another Equipment Unit may use electric heating coils.

Different consumer and producer considerations

The following are some points to consider when building Formula/Report data structures for different consumers and producers of data.

If data is . . .	Then, consider . . .
Consumer/Produced at same level as defined.	Parameter data used at the parent level to define different logic paths in execution (for example take Path A versus Path B), and the data is not sent to lower level components.
	Report data that is result of a expression evaluation (for example through a Step Output expression that collects data from multiple CMs) and used to define the successful completion or need to re-execute the logic path in execution.
Consumed at different levels than defined.	Parameters data used to define amounts of material to be added, defined at top level, and sent down through the levels as needed to point where child component consuming data is executing.
	Report data capturing the temperature reached during a heating function, collected at lower level, and sent up through the levels as needed to point where the parent component capturing data is executing.

11.2 Considerations for Phase block types

Sharing Phase types across recipe layers

Formula and report parameter definition can be shared between recipe layers through Phase blocks and Data blocks. The Data block of a child Recipe or an SCM must use the same Phase type as the linked Phase block in the parent Recipe. This is to utilize the benefit of automatic alignment of formula and report parameters.

Restrictions for changing Phase block types

Refer to the topic “Considerations for modifying an equipment model” on page 62.

11.3 Considerations for enumeration sets

Choosing between private and public enumeration sets

You can choose to create a private enumeration set or a public enumeration set based on how the enumeration sets are likely to be used. If an enumeration set needs to be referenced in several Phase block types, you can create a public enumeration set. However, once referenced, any modifications to a public enumeration set are restricted. Once referenced, you cannot make any modifications to the existing enumeration names, values, or change the order. You can only add new values at the end of the list. Hence, it would be good to create public enumeration sets if they are likely to be used in several Phase block types and if they are not likely to be modified often.

With private enumeration sets, there are fewer restrictions on the modifications since the scope is limited to the Phase block type in which they are defined. This is applicable only if the Phase block instance is not used in any strategy. If there are any instances of the Phase types in the strategies, modifications to the existing private enumeration names, values, and order are restricted.

Also, note that private enumeration sets are automatically exported and imported with the associated Phase block types. However, public enumeration sets need to be exported and imported manually along with the Phase block types that reference them.

For example, in the Yogurt production example, you can create private enumeration sets for XFEROUT Phase block type. You can create a public enumeration set for homogenizing the milk since it can be referenced in any Phase block type that is used for making Yogurt.

11.4 Differences between a UCM and a Unit class

Unit classes are user-defined templates of a UCM. Unit instances are identical to UCMs. Although Unit classes are used for Class-based recipes and UCMs for instance-based recipes, it is recommended to use Unit instances derived from Unit classes even for instance-based recipes. This would enable greater flexibility and help in future expansions.

11.5 Usage of Map blocks

Map block types are used for defining the functions of a Unit class. Here is a list of points on how Map blocks can be useful for Class-based recipes.

- Map blocks can be reused across multiple Unit classes/UCMs. For example, if you have two different Unit classes performing similar functions, you can reuse the Map block types across the Unit classes.
- A Unit class can contain multiple Map blocks.
- Map blocks support multiple types of references: function, block, and parameter references. To learn more about these references supported by Map blocks and how to use them, refer to the "Build equipment model" chapter in the *Batch Implementation Guide*.

11.5.1 Considerations for creating functions

- UDT reference is used as a block reference type when the data to be accessed through the reference can be resolved for every Unit instance through a derivation child of the specified UDT. A function reference must contain a data block that matches the associated function type.
- An instance that has the specified reference type in its derivation hierarchy can be used as a function type. For example, consider the following derivation hierarchy.

  PHCDB (block type)

  PH_UDT1 (UDT derived from PHCDB)

  PH_UDT1_1 (UDT derived from PH_UDT1)

  Consider the following:

  – SCM1 data block is derived from PHCDB.

  – SCM2 data block is derived from PH_UDT1.

  – SCM3 data block is derived from PH_UDT1_1.

  If the function reference type is specified to be PHCDB, all SCMs that are configured with the PHCDB are valid function references. However, if a function reference type is specified as PH_UDT1, SCM1 is not a valid function reference.

11.5.2 Considerations for creating block references

- UDT reference is used as a block reference type when the data to be accessed can be resolved for every Unit instance through a derivation child of the specified UDT. This provides access at runtime to a derivation child of the UDT through "UNIT[X].MAP.ReferenceName.param," or "UNIT[X].MAP.ReferenceName.bb.parameter."

  Attention

  Name.bb.parameter is allowed only when bb is derived from a block type that resides in an Experion CEE.

- A standard or a custom block type reference is used when the data to be accessed through the reference can be resolved for every Unit instance through a block of the specified type. This provides access at runtime to a derivation child of the block type through "UNIT[X].MAP.ReferenceName.param."

  – For tagged block types: A TAGNAME is specified for the block reference.

  – For basic block types: A TAGNAME.bb is specified for the block reference.

- An instance that has the specified reference type in its derivation hierarchy can be used as a block reference type. For example, consider the following derivation hierarchy.

  REGCTL: PID (block type)

  PID_UDT1 (UDT derived from PID)

  PH_UDT1_1 (UDT derived from PID_UDT1)

Consider that the CM1 has PID, CM2 has PID_UDT1, and CM3 has PH_UDT1_1. If the block reference type is specified to be REGCTL: PID, all the CMs (CM1, CM2, and CM3) that are configured with the PID block are valid block references. However, if a block reference type is specified as PID_UDT1, CM1 is not a valid block reference.

11.6 Considerations for modifying an equipment model

Modifications to the equipment model can be restricted based on how the Unit class or type is used.

Extending the capability of a Unit class is always supported. This includes the following:

- Adding new parameters to a Phase type that is used by a Map block type function.
- Extending the capability of a Map block type used by a Unit class. This includes adding new functions, new block references, and new parameter references.
- Adding new Map blocks or other basics blocks to a Unit class.

11.6.1 Restrictions for modifying enumeration sets

The following modifications cannot be made if an enumeration set is referenced/used in any block type/UDT or any other strategies.

- The existing enumeration set (private and public) cannot be deleted.
- The enumeration set ordinal values cannot be modified.

However, a new enumeration set member (enumeration string and enumeration ordinal) can be added to an existing enumeration set, even if the enumeration set is used in any block type/UDT or any other strategies. Enumeration string can be modified even if the enumeration set is used in any block type/UDT or any other strategies.

A public enumeration holder cannot be deleted from the library view if it contains configured enumeration sets.

11.6.2 Considerations for modifying Phase block types

- Phase block types cannot be renamed after they have been used by a Map block type as a Phase type of a function or a block type of a block reference.
- Phase block types cannot be deleted after they have been used by a Map block type as a Phase type of a function or a block type of a block reference, or used as a Data block or a Phase block or a recipe.
- A parameter may not be deleted, renamed or its type be modified from a Phase block type after the Phase block type has been used as a Data block or a Phase block or a recipe.
- If an enumeration is used in a formula or report parameter and any Phase block instance exists, then the enumeration set members cannot be deleted, ordinals, and values cannot be changed, and the name of the enumeration set cannot be changed.

11.6.3 Considerations for modifying Map block types

- A Map block cannot be deleted once it has been contained in a Unit class or a UCM.
- A function, block reference, or a parameter reference may not be deleted from a Map block type once the Map block has been contained in a Unit class or a UCM.
- A function name or a Phase type cannot be modified in a Map block type once the Map block has been contained in a Unit class or a UCM.
- A block reference name or a block type cannot be modified in a Map block type once the Map block has been contained in a Unit class or a UCM.
- A Parameter reference name or a type cannot be modified in a Map block type once the Map block has been contained in a Unit class or a UCM.
- UDTs cannot be renamed once they have been used by a Map block type as a block type of a block reference.
- UDTs cannot be deleted once they have been used by a Map block type as a block type of a block reference.

11.6.4 Considerations for modifying Unit classes and UCMs

Considerations for modifying Unit classes

Basic blocks (Map blocks and other basic blocks) that are contained by a Unit class cannot be deleted if the Unit class or the derivation children of the Unit class is associated with a Master Recipe and the Master Recipe is loaded. Basic blocks can be deleted from a Unit class if the Unit class or derivation children of the Unit class is not associated with a Master Recipe, or if no associated Master Recipes are loaded. Be cautious in deleting blocks when the Unit class or derivation children of the Unit class are associated with unloaded Master Recipes, as this could cause the Master Recipe to become invalid.

Considerations for modifying UCMs

Basic blocks (Map blocks and other basic blocks) that are contained by a UCM cannot be deleted if the UCM has been associated with a Master Recipe and the Master Recipe has been loaded. Basic blocks can be deleted from a UCM if the UCM is not associated with a Master Recipe, or if no associated Master Recipes are loaded. Be cautious in deleting blocks when the UCM is associated with unloaded Master Recipes, as this could cause the Master Recipe to become invalid. You may search for Master Recipes that have been associated with a Unit class or a UCM and/or refer to the "Validate block references" topic in the *Batch Implementation Guide*.

Reloading UCMs with new reference values while a Control Recipe is executing against a UCM

You may modify the values of a UCM and reload it while the UCM is selected for an executing Control Recipe. The following list explains how the changes impact the Control Recipe.

- Changes to a function reference will not affect a Phase block that has already started or completed execution. A Phase that has not started execution, uses the new value of the function reference.
- Changes to block references and parameter references affect the steps and transitions, while the step or transition is executing the next time.

12 Guidelines/considerations for building recipes

Related topics

"Differences between instance-based recipes and Class-based recipes" on page 66
"Acquisition of resources support" on page 67
"Alias table and user-defined templates" on page 69
"UCM or equipment name versus alias reference usage" on page 71
"Memory utilization and loading considerations" on page 72
"Using an existing RCM or an SCM in Class-based recipes" on page 73
"Using an RCM or SCM configured to use HISTVALUE and RECTARGET as a child of a Master Recipe/ Control Recipe" on page 76

12.1 Differences between instance-based recipes and Class-based recipes

The following table illustrates some of the differences between instance-based recipes and Class-based recipes.

Instance-based recipes	Class-based recipes
Designed for specific units and can be run against only a specific Unit instance.	Designed for Unit classes and can be run against any unit in a Unit class.
Complex to maintain since every recipe needs to be configured as part of recipe building.	Simple to configure and maintain since only Master Recipes needs to be configured during recipe building. Control Recipes can be instantiated during run time.
Only one instance of the recipe can be run at any point of time.	Multiple instances of the recipes can be run simultaneously on different unit instances.
Does not support reuse since recipes are equipment-specific. However, UDTs can be used for creating recipe templates.	Supports reuse since recipes are not equipment-specific.
Direct references to functions and parameters must be provided during recipe configuration.	References to functions and parameters are provided through Map blocks during Master Recipe configuration. These references are resolved during Control Recipe execution.

12.2 Acquisition of resources support

You can separate batch control into the following two areas.

- Sequence/Device Control which consist of the actual manipulation of control devices, including normal execution as well as abnormal execution through exception handlers. The development of control strategies is completed by a Control Engineer that has an intimate knowledge of the control actions and equipment/ devices used to achieve a defined goal, and are implemented in SCM/CM.
- Recipe/Procedure Control which consists of the orchestration of control functions, rather than the actual manipulation of control devices, including the normal execution in as well as how to deal with abnormal execution being reported/propagate from control functions. The development of recipes is completed by a Formulation/Recipe Engineer that has intimate knowledge of the production of a product. For example, the order of ingredient execution, but not necessarily the details of how this has been accomplished.

By design, this clear separation between "functions" and "how the functions are achieved" allows for the construction of a batch automation solution using the modular batch automation concepts. To achieve a defined manufacturing goal that is implemented in a *Recipe*, the *Recipe* author must have a detailed knowledge of the initiation and the co-ordination of control functions. Likewise, the SCM author must have a detailed knowledge of how the initiation and the co-ordination of control devices are, to achieve the goals on the associated function.

Due to the dynamic nature of batch control, acquisition is key capability required for effective and secure operations and is common across these two areas. Commonly, in batch plants, multiple independent components are needed to complete a control action, but it is important to guarantee that the components are used exclusively for a single batch and not used in the execution of a different batch simultaneously.

12.2.1 Acquisition support in EBM

EBM supports acquisition in the following ways.

- Unit acquisition by a *Recipe* at the time when it is needed to support the execution of a batch/recipe that coordinates control function to make a specific product (or complete a specific task in the case of Procedural Operations). Example of this is a Mixer that can be used for executing multiple recipes (RedDye, BlueDye), but it is clear that it is necessary to only execute a single recipe on the Mixer at any time (mixing dyes is not desirable). To achieve this, a *recipe* makes a request to "acquire" the unit, complete the recipe, and then "release" the unit.
- Child *Recipe*/SCM acquisition by a parent *Recipe* at the time when a child component is needed to complete a defined function. The child *recipe*/SCM executes its defined task, and then is released.
- CM acquisition by an SCM at the time when it is needed to support a control action. The CM executes a defined action, and then is released. What is unique about control devices is that multiple SCMs can request and effectively use the CM simultaneously. This introduced the concept of non-exclusive use.

Note

For more information on unit acquisition support for Class-based recipes, refer to the *Batch Implementation Guide*.

12.2.2 Parent and child recipes

A *Recipe* has the ability the run either as a parent or a child. When running as a parent, the *Recipe* is responsible for acquiring any UCM as needed, and initiating child components as needed. The key decision when initiating a child *Recipe* is how the child *Recipe* should deal with UCM acquisition. This option is explicitly defined through the Phase block through which the child *Recipe* is initiated.

- Initiate the child *Recipe* and instruct the child component to execute the UCM acquisition logic. This is the case, where the parent does not acquire the UCM needed by the child, but requests the Child component to acquire UCM as configured. In some cases, the child component is not configured to acquire, whereas in other cases it is. The parent *Recipe* will delegate the acquisition of UCM (if configured) to the child *Recipe* .

- Initiate the child *Recipe* and do not instruct the child components to execute the UCM acquisition logic. This is the case where the parent has acquired the UCM needed by the child, and therefore does not request the child to acquire UCM. The child component will execute but will expect that the parent component has acquired a UCM needed for execution.

  If the parent acquires an UCM, a child of the *Recipe* (or child of the child of the *Recipe*) should not attempt to acquire the same UCM (while the parent has acquired the UCM) because it will cause a deadlock condition.

This flexibility allows a single *Recipe* to be configured in such a way that it can execute as either a parent or child component, and each component (child *Recipe* and parent *Recipe*) can decide what component is responsible for acquiring UCM as needed. For example:

- *Recipe*A is configured to acquire UCMA, and this is an example of a *Recipe* that requires exclusive access to the equipment associated with an UCM. When running *Recipe*A as a child *Recipe*, the parent may have acquired UCMA on behalf of the child, hence, *Recipe*A will not acquire UCMA. But when running as a parent, *Recipe*A will acquire UCMA as configured. This is important, since *Recipe*A action must only execute on an equipment unit, and therefore acquisition of UCMA is needed either directly or indirectly (through parent acquisition of UCMA).
- *Recipe*A is not configured to acquire UCM, and this is an example of a *Recipe* that does not require exclusive access to any equipment. This is often the case when the *Recipe* is "task" based versus "product" based. Task-based function often can run in parallel with multiple batches, whereas product-based function are used directly in support in the making of a specific batch and therefore must guarantee exclusive access to equipment through UCM acquisition. Procedural Operations often define task-based functions where the end goal is to complete a specific function (switching over from Primary PumpA to Backup PumpB), whereas in batch automation often define product-based functions where end goal is to produce a specific product (making 1000# of Red Dye #5).

When dealing with acquisition for resources, it is expected that there are times when multiple requesters are requesting for the same components simultaneously. In this case, acquisition needs to support queuing of requests so that an orderly management of multiple requesters is achieved. For example, components are acquired in the order in which they are requested.

12.3 Alias table and user-defined templates

Recipes are configured to achieve a specific and defined function/task, and a single recipe can execute on multiple units that are capable of making the same products. For example, in a Fine Chemicals plant there are often multiple trains of Equipment/Devices that run in parallel, each executing a single batch/recipe to make a "Lot/Run" of a product. The finished products from a recipe executing on separate units are combined into common finished product tanks. Hence, this single recipe is executed against each separate unit. The changes to recipes are done once and then it is necessary to run this action on multiple units. To reduce the engineering efforts, (and to assure consistency between these multiple instances), templates are used to define the recipe, and the multiple instances are generated from this single template and loaded into defined CEE, as illustrated in the following figure.

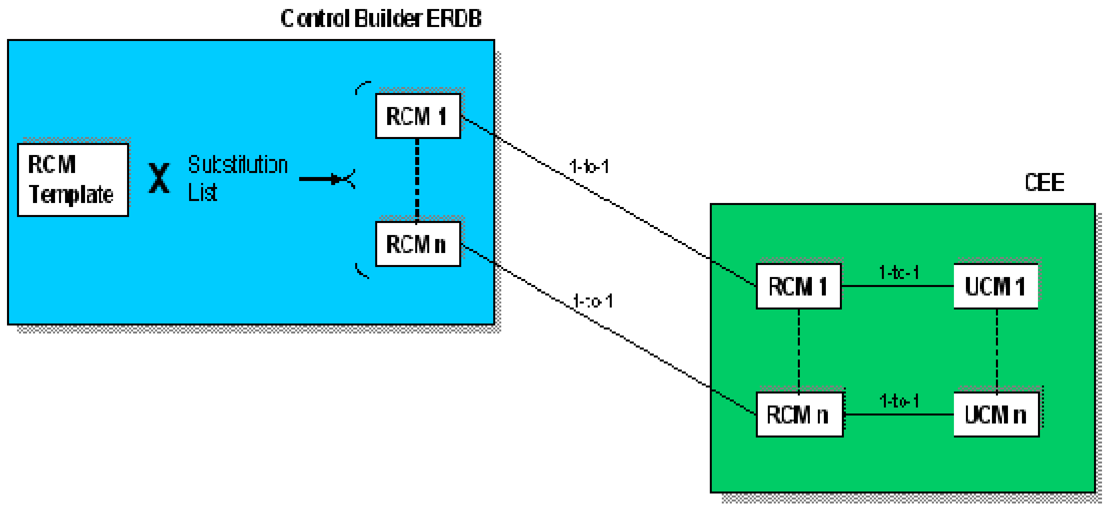


Figure 27: Creating recipes using alias table and user-defined templates

The other case that is often encountered is where a recipe can be shared across multiple units, and needs to change the child RCM/SCM it coordinates based upon different lineups. In this case, there is one recipe but it requires a runtime level of re-configuration based upon the equipment used. Alias table can support a recipe that needs to change the child components initiated, but the logic to complete the orchestration is the same. For example, in a Consumer Product plant there can be a single piece of equipment that is used to finish a product, and it can support as input intermediate product from multiple upstream equipment. To support this, a recipe needs to re-configure itself based upon the defined plant configuration, and this must be accomplished at runtime (no Engineer load action is possible). Alias table allows for specification of different configuration, and the selection of the defined configuration (instance selection) can be achieved at runtime.

EBM is unique in that it allows for the combination of capabilities to support flexible manufacturing plants. An example would be where a single finishing equipment unit can support multiple mixing trains (4 Mixers to 1 Finishing Unit) where we need alias tables to provide the dynamic runtime indirection capabilities, yet also require UDTs to generate recipes across multiple units that can achieve the same task.

Attention

- You can also achieve the same capability of specifying different configuration and selecting the defined configuration (selecting the instance) at runtime by using Class-based recipes. Master Recipes do not support the use of Alias Tables.
- You can modify the active loadable parameter values and load them while active without having to inactivate the strategy or set the CEE to Idle. In addition, the **Load Alias Table** menu option is renamed as **Load Values while Active** as part of this enhancement. Refer to the *Control Builder Components Theory* and *Control Building User's Guide* for more information about this option.
- There are no specific guidelines or automatic ways of converting an RCM with alias table to a Class-based recipe. You need to manually perform this task.
- If an alias name or an alias reference configured in an SCM/RCM referring to a CM then you can navigate to the SCM from the **Cross References** panel of the CM.

12.4 UCM or equipment name versus alias reference usage

As discussed previously, resource acquisition is an important concept in batch automation. The decision of which UCM to acquire is most often achieved through "direct" configuration in the RCM. Instance-based RCMs are configured to execute against a single UCM and this static definition is defined prior to loading the RCM to the CEE.

- Single UCM to acquire is known at configuration time
- Equipment name is configured through a direct specification in RCM
- RCM is loaded with fixed direct reference
- RCM executes against the defined static UCM

But in other cases, a single RCM has the ability to run against multiple units, and which UCM to acquire is not known until runtime. In this case, the decision is deferred until runtime and is achieved through "indirect" runtime selection from a group of configurations defined through an alias table.

- UCM(s) to acquire is known at configuration time
- Configured through alias table in RCM
- Equipment name is configured through indirect reference through alias table in RCM
- RCM is loaded with dynamic indirect reference (through alias table)
- Instance of equipment unit to execute against is selected at runtime
- RCM executes against the dynamically selected UCM (through alias table)

Note

"UCM or equipment name versus alias reference usage" applies only to RCMs. It does not apply to Class-based recipes.

12.5 Memory utilization and loading considerations

Considerations for number of Control Recipes and RCMs running simultaneously in a controller

To determine how many Control Recipes and RCMs you can run simultaneously in a controller, refer to the following specifications and work with your Honeywell representative.

- *Experion Networks and CEE Controller Capacities and Specifications*
- *EBM Specifications*

The MAXOWNERS of MR parameter can be used for configuring the number of Control Recipes that can be executed for a Master Recipe. The maximum number of Control Recipes that can be executed for a Master Recipe is 20. Additionally, the following are some of the diagnostic parameters that help in determining the memory usage of recipes.

- When the recipes are executing, you can refer to the **CPU Overruns** and **Batch** tabs on the controller's CEE configuration form to monitor memory allocation/utilization, and overruns. Note that the **CPU Overruns** tab contains only the RCM/SCM overrun information. Use the **Batch** tab for Control Recipe overruns information.
- For Master Recipes and Control Recipes, in addition to the tabs mentioned on the CEE configuration form, you can also refer to the following diagnostic parameters on the **Status** tab of the Master Recipe configuration form.
  - Sum of used memory (**CONTAINMEM**): Indicates the memory allocated for the Master Recipe and all contained component blocks (not counting the contained containers).
  - Control Recipe (**MEM1CR**): Indicates the maximum memory demand of one Control Recipe.
  - Curr. Control (**MEMALLCR**): Indicates the memory currently used by all Control Recipes for the Master Recipe. When no Control Recipes are executing, this value is zero. This value may some times reach "n" times MEM1CR, where "n" is the number of executing Control Recipes.

Note
Refer to the *Control Builder Parameter Reference* for more information on these parameters.

Determining number of activities that can be loaded to a controller

For activities, memory usage happens in two phases.

- When the CEE is loaded. In this scenario, the Number of Activities (NUMACT) parameter determines the number of activities that are loaded to the controller. Each activity loaded to the CEE consumes some Memory Units (MU). Hence, calculate the memory consumption for loaded activities.
- When a Data block is loaded to a manually-created activity (during the initialization and pre-execution stages of the activity). This is determined by the size of the Data block. The size of a Data block is determined based on the number of integer, boolean, float, and string parameters stored in the Data block.

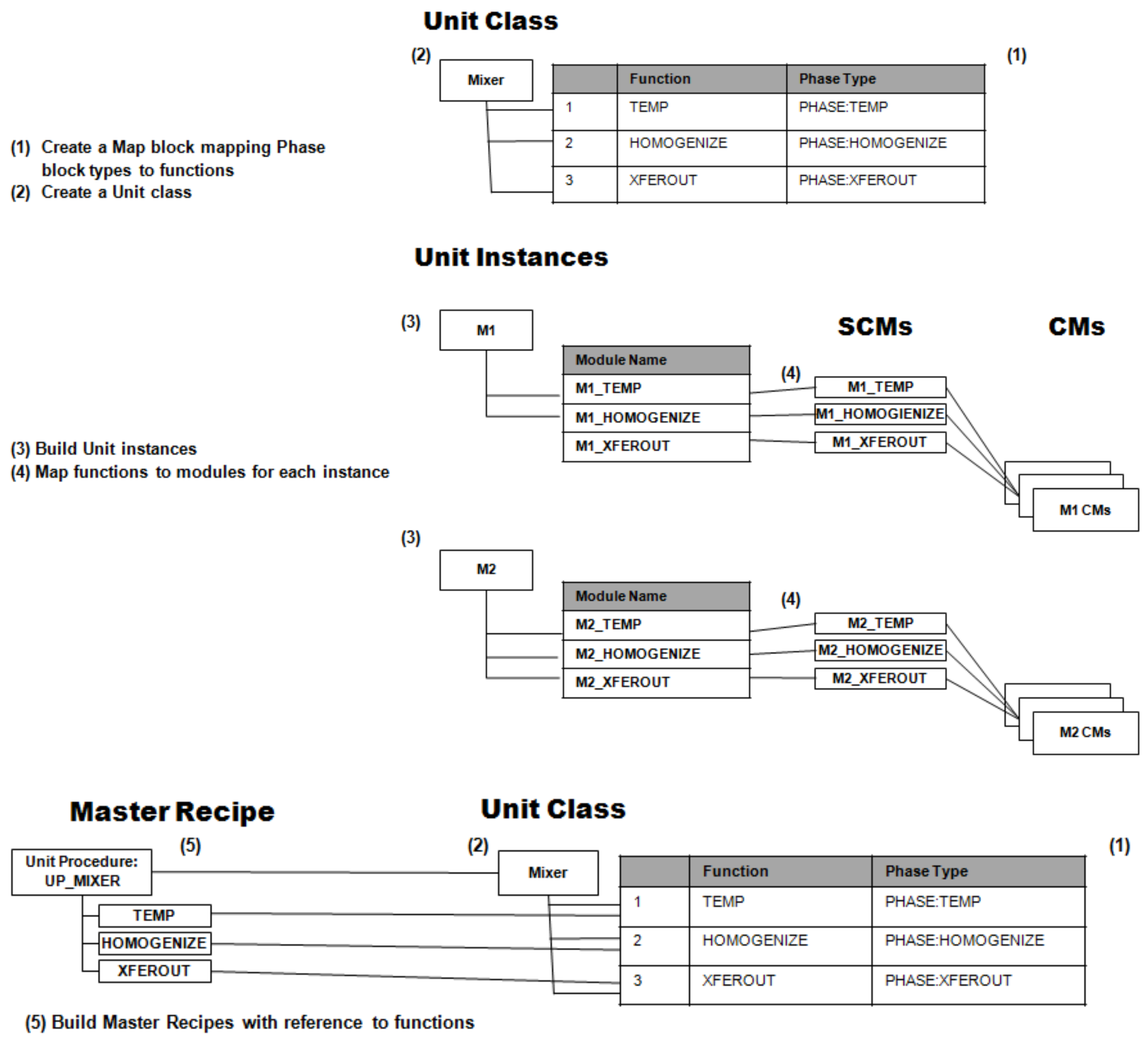
Note that the memory consumption numbers is not the same for all controllers. However, there is only a slight variation between the all the controllers. Refer to the *Experion Networks and CEE Controller Capacities and Specifications*, *EBM specifications*, or contact your Honeywell representative for more information.

12.6 Using an existing RCM or an SCM in Class-based recipes

An RCM or an SCM which is created with an Experion release earlier than R410 can be used in a Master Recipe in one of the following two ways.

- By linking the RCM or the SCM to the Master Recipe through a Map block function.
- By adding the RCM or the SCM directly to a Master Recipe. In this case, there would be no mapping of blocks to the Unit instance running the Master Recipe. A CM or any other block referenced by the RCM or the SCM would be common to all the units.

Refer to the following illustrations.



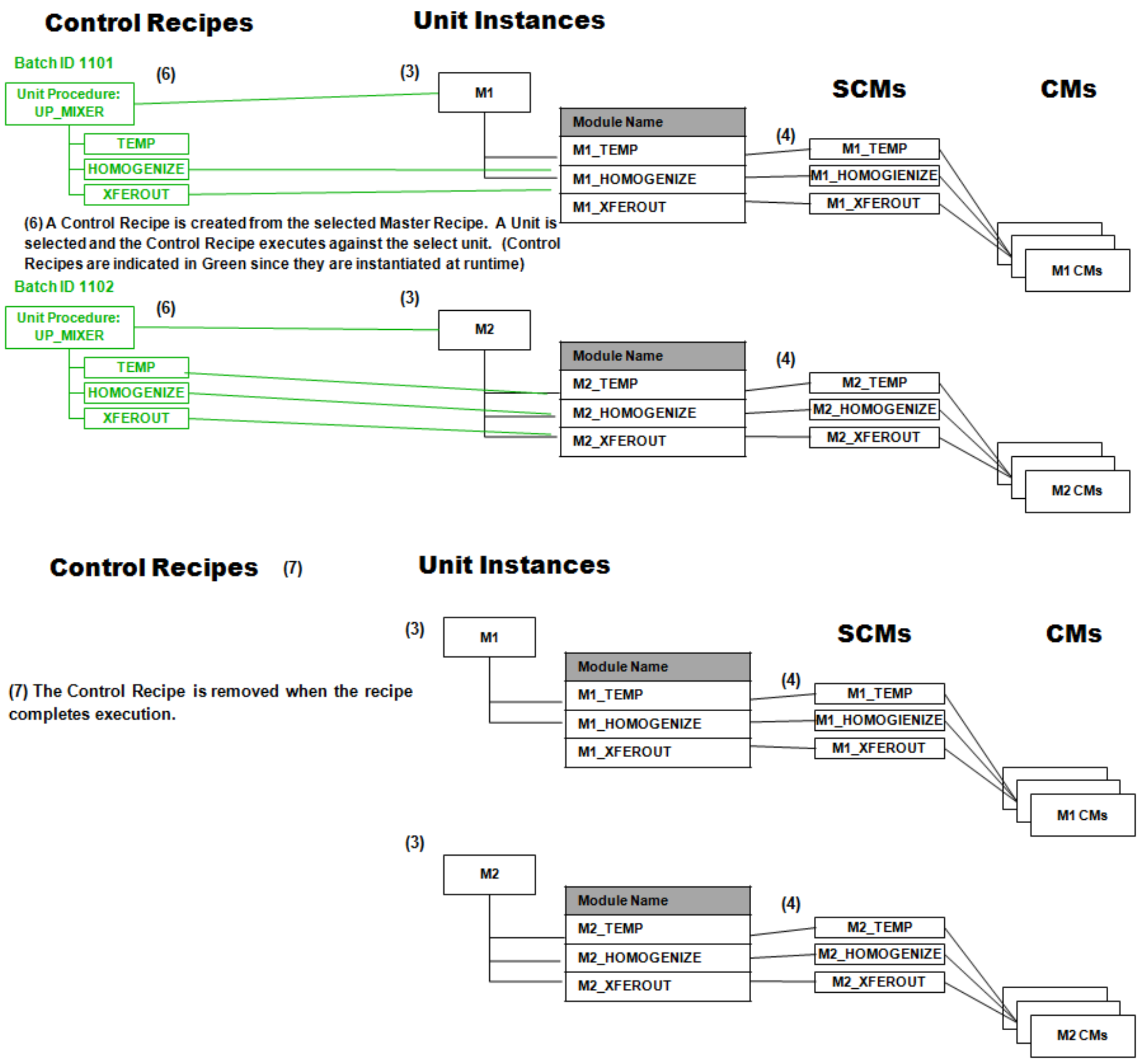

Control Recipes
Unit Instances
Batch ID 1101
(6)
(3)
M1
SCMs
CMs
Unit Procedure: UP_MIXER
Module Name
(4)
TEMP
HOMOGENIZE
XFEROUT
M1_TEMP
M1_HOMOGENIZE
M1_XFEROUT
M1_HOMOGIENIZE
(6) A Control Recipe is created from the selected Master Recipe. A Unit is selected and the Control Recipe executes against the select unit. (Control Recipes are indicated in Green since they are instantiated at runtime)
M1 CMs
Batch ID 1102
M2
M2_TEMP
M2_HOMOGENIZE
M2_XFEROUT
M2 CMs
Control Recipes (7)
(7) The Control Recipe is removed when the recipe completes execution.

Figure 28: Linking an RCM or an SCM to a Master Recipe through a Map block function

Note
The numbers in this illustration refer to the configuration steps.

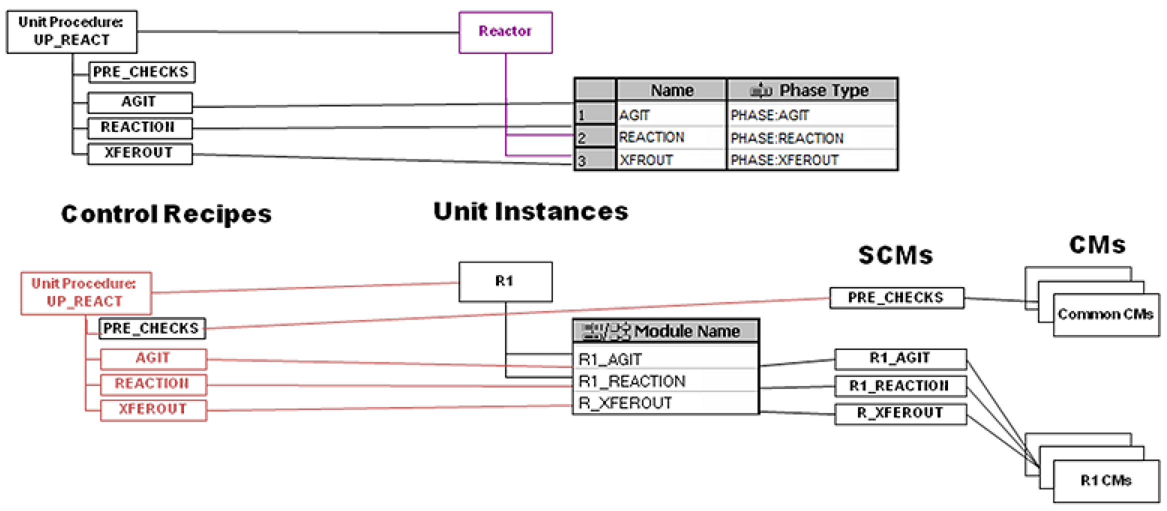


Figure 29: Adding an RCM or an SCM directly to a Master Recipe

The SCM PRE_CHECKS is added to the Master Recipe. Unlike the other Phases in the recipe, it is executed by the Control Recipe without any Unit instance mapping.

Note

Additionally, you can refer to the "Building Master Recipes" chapter in the *Batch Implementation Guide* for more information about how to add different recipe layers in a Master Recipe.

12.7 Using an RCM or SCM configured to use HISTVALUE and RECTARGET as a child of a Master Recipe/Control Recipe

Creating a layered Master Recipe is most beneficial when a Phase block in a parent matches the Data block of the child recipe or SCM. However, some RCMs or SCMs may have already been configured to use HISTVALUE and RECTARGET parameters. These RCMs and SCMs may be used as children with Master Recipe as parents. But, the following restrictions apply.

- Manual configuration of the Phase block in the parent Master Recipe is required.
- If the child RCM or SCM is selected at runtime through a Unit map, all children must be of the same type. There cannot be a mix of two units with SCM children and a third unit with an RCM child. Also, all children must be configured to use HISTVALUE and RECTARGET parameters.

Note

- In this scenario, if you are using RECTARGET, be aware that the parameters do not appear on the appropriate tabs of the summary displays.
- For information on configuration of this scenario, refer to the "Building Master Recipes" chapter of the *Batch Implementation Guide*.

13 Guidelines/considerations for building custom graphics for Class-based recipes

The following are some of the guidelines/considerations for building custom graphics for Class-based recipes.

- The Procedure and Sequence toolkit controls have been developed to bind only to SCM/RCM/UCM and Activity points and their sub-elements. They cannot be bound to Master Recipes. However, they can be bound to Control Recipes at runtime.
- The push button can be configured to invoke **Create Activity** dialog box with filters defined at build time.
- The **Activity Table** can configured specifically for batch only, procedural operations only, or a mixture of both.

Note

For more information on how to build custom graphics, refer to the *Procedure and Sequence Custom Display Building Guide*.

14 About import/export support for Class-based recipes

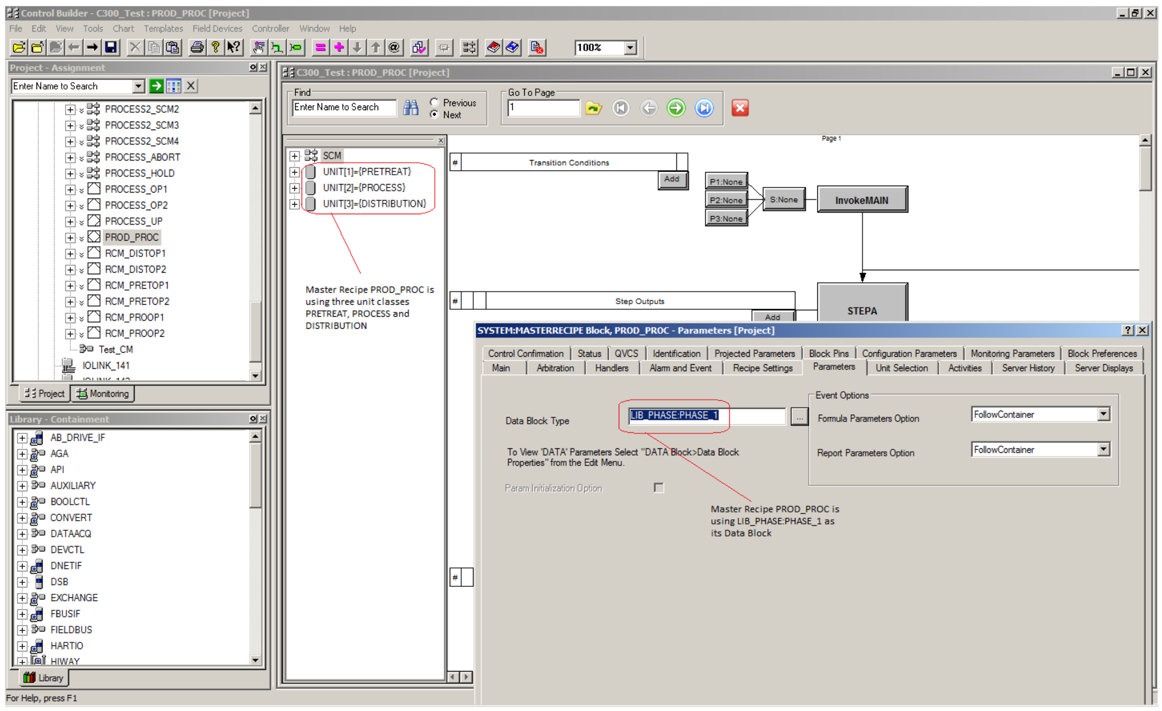
Import and export functionality available in Control Builder is supported for all the components of Class-based recipes. These include block types, templates, and strategies. Block types include Public enumeration holders, Phase block types, and Map block types. Templates include Unit classes or UDTs created for Phase block types or *Recipes*/SCMs. Strategies include instance of a recipe, SCM, Unit instance, or a Master Recipe. If configured, Class-based recipe strategies also include intercluster peer-to-peer blocks – cluster, node, and proxy tag (proxy SCM, proxy RCM, and proxy Master Recipe).

About component dependencies

Class-based recipes system components are dependent on one or more of the other components to achieve the necessary functionality. For example, a Master Recipe using a Unit class or a Master Recipe using a Phase block as it's Data block. Such scenarios lead to dependency of components. If a Master recipe is using a Unit class, the Master Recipe has a dependency on that Unit class. Similarly, if it is using multiple Unit classes, it would have a dependency on all those Unit classes.

You should know the component dependencies before exporting and importing. To successfully import a component, it is important to ensure that its dependencies can be resolved during import. This can be achieved either by importing the dependent components also or by ensuring that dependent components are already present in the Control Builder before importing the component.

The following illustration depicts a Master Recipe PROD_PROC that is using three Unit classes - PRETREAT, PROCESS, and DISTRIBUTION. Apart from that, it is also using LIB_PHASE:PHASE_1 as its Data Block. Hence, the Master Recipe PROD_PROC is dependent on the Unit classes namely - PRETREAT, PROCESS, DISTRIBUTION, and Phase block type LIB_PHASE:PHASE_1.



Master Recipe PROD_PROC is using three unit classes PRETREAT, PROCESS and DISTRIBUTION

Data Block Type: LIB_PHASE:PHASE_1

To View 'DATA' Parameters Select "DATA Block>Data Block Properties" from the Edit Menu.

Formula Parameters Option: FollowContainer

Report Parameters Option: FollowContainer

Master Recipe PROD_PROC is using LIB_PHASE:PHASE_1 as its Data Block

Figure 30: An example Master Recipe that is using three Unit classes

Attention

If you export the Master Recipe, Unit instances, and Control Modules, all the dependent library blocks such as Unit class, Map block types, Phase block type, and Custom blocks are exported. However, if you export only the Unit class, the Unit class and the Map block types are exported. Therefore, you must manually export the Phase block types and Custom blocks.

Component dependencies can also be multilevel caused by indirect references of the dependent components. In the following example, Master Recipe 'DIST_OP1' is dependent on the following:

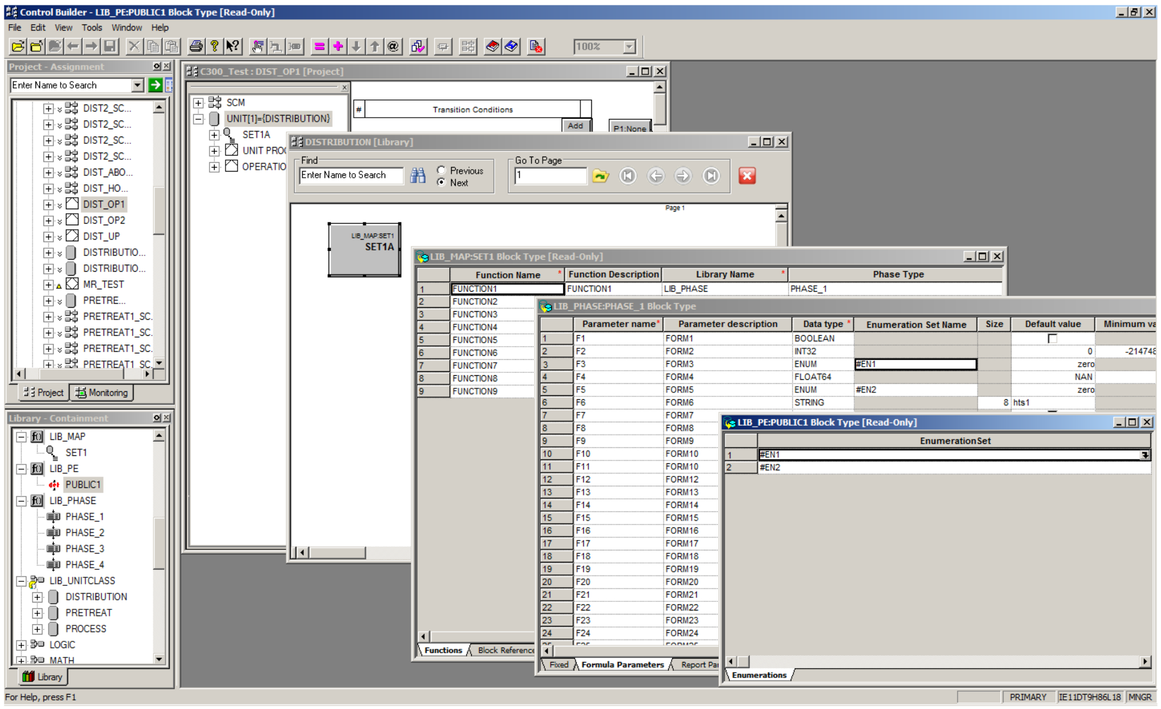
- On Unit class named 'Distribution' because it is using it.
- On Map block type 'LIB_MAP:SET1' because Unit class 'DISTRIBUTION' is having an instance of Map block 'LIB_MAP:SET1.'
- On Phase block type 'LIB_PHASE: PHASE_1' because Map block 'LIB_MAP:SET1' has a function named FUNCTION1 that has Phase type as 'LIB_PHASE:PHASE_1'.
- On Public enumeration holder block 'LIB_PE:PUBLIC1' because Phase block type 'LIB_PHASE:FUNCTION1' has a parameter named F3 and F5 of enumeration type that are using enumeration sets name #EN1 and #EN2. Enumeration sets #EN1 and #EN2 are defined in Public enumeration holder block 'LIB_PE:PUBLIC1'.

You can also use Experion Search utility to find the dependencies of a component. For more information about search support for batch objects, refer to the topic "Searching for batch scenarios".

Attention

Public enumeration sets are not exported automatically. They need to be explicitly exported while exporting a Phase block type.

The following figure illustrates the export and import of the Master Recipe 'DIST_OP1'.



LIB_MAP:SET1 Block Type [Read-Only]

	Function Name	Function Description	Library Name	Phase Type
1	FUNCTION1	FUNCTION1	LIB_PHASE	PHASE_1
2	FUNCTION2			
3	FUNCTION3			
4	FUNCTION4			
5	FUNCTION5			
6	FUNCTION6			
7	FUNCTION7			
8	FUNCTION8			
9	FUNCTION9			

LIB_PHASE:PHASE_1 Block Type

	Parameter name	Parameter description	Data type	Enumeration Set Name	Size	Default value	Minimum va
1	F1	FORM1	BOOLEAN			☐	
2	F2	FORM2	INT32			0	-214748
3	F3	FORM3	ENUM	#EN1		zero	
4	F4	FORM4	FLOAT64			NAN	
5	F5	FORM5	ENUM	#EN2		zero	
6	F6	FORM6	STRING		8	hts1	
7	F7	FORM7					
8	F8	FORM8					
9	F9	FORM9					
10	F10	FORM10					
11	F11	FORM10					
12	F12	FORM12					
13	F13	FORM13					
14	F14	FORM14					
15	F15	FORM15					
16	F16	FORM16					
17	F17	FORM17					
18	F18	FORM18					
19	F19	FORM19					
20	F20	FORM20					
21	F21	FORM21					
22	F22	FORM22					
23	F23	FORM23					
24	F24	FORM24					

LIB_PE:PUBLIC1 Block Type [Read-Only]

	EnumerationSet
1	#EN1
2	#EN2

Figure 31: An example for export and import of a Master Recipe

Note

For information about how to import/export Class-based recipes, refer to the "Maintenance" chapter in the *Batch Implementation Guide*.

About import/export of Public enumeration holders

Public enumeration holder blocks can be exported or imported using the Import/Export functionality available in Control Builder. A public enumeration holder block can be exported or imported independently as it does not have a dependency on other components. However, the public enumeration sets and the associated Phase block types must be imported together.

About import/export of a Phase block type with parameter type Public enumeration

Phase block types can be exported or imported using the Import/Export functionality available in Control Builder. A Phase block type can have a dependency on the Public enumeration holder block if the Phase block type has one or more parameters of type Public enumerations. To successfully import the Phase block type, either the Public enumeration holder block needs to be imported along with Phase block type or it needs to be available in the Control Builder library before importing the Phase block type.

Attention

Public enumeration sets are not exported automatically. They need to be explicitly exported while exporting a Phase block type.

14.1 About import/export of Map block types

Map block types can be exported or imported using the Import/Export functionality available in Control Builder. A Map block type can have a dependency on user-defined block types like Phase block types or Custom Data block types. A Map block type has a dependency on a Phase block type if the Map block type references it as a type in the definition of a function reference. Similarly, Map block type has a dependency on a Custom Data block type if a Map block type references it as a type in the definition of a block reference. Also, the block reference added using a standard block in the Control Builder library does not add a dependency because it is present by default. For example: block reference to 'REGCTL:PID' does not add a dependency.

For a Map block type, dependencies can also be multilevel caused indirectly by dependencies of components on which it is dependent on other components. For example, if a Map block type depends on a Phase block type and that in turn uses public enumeration holder block, the Map block type has a dependency on that public enumeration holder block too.

14.2 About import/export of Unit classes

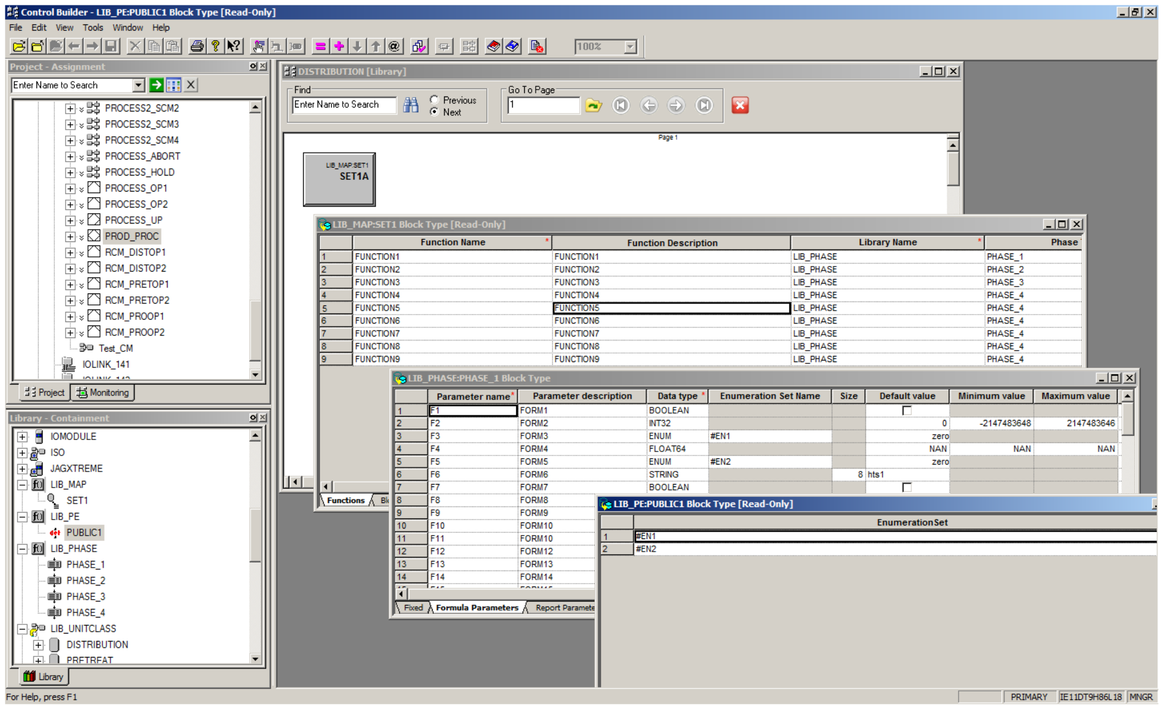
Unit class can be exported or imported using the Import/Export functionality available in Control Builder. A Unit class can have a dependency on user-defined block types, such as a Map block type or a Custom Data block type. A Map block type has a dependency on a Phase block type if the Map block type references it as a type in definition of function reference. Similarly, a Map block type has a dependency on a custom Data block type if the Map block type references it as a type in definition of block reference.

For a Unit class, dependencies can also be multilevel caused indirectly by dependencies of components on which it is dependent. For example, assume that a Unit class contains a Map block type that depends on a Phase block type. In this scenario, if the Phase block type used in the Map block type uses a public enumeration holder block then the Map block type has a dependency on public enumeration holder block.

In the following example, the Unit class ‘Distribution’ is dependent on the following:

- On Map block type 'LIB_MAP:SET1' because Unit class 'DISTRIBUTION' has an instance of Map block 'LIB_MAP:SET1.'
- On Phase block type 'LIB_PHASE: PHASE_1' because Map block 'LIB_MAP:SET1' has a function named FUNCTION1 that has Phase Type as ‘LIB_PHASE:PHASE_1’. Similarly, it has a dependency on Phase block types 'LIB_PHASE: PHASE_2,' 'LIB_PHASE: PHASE_3,' and 'LIB_PHASE: PHASE_4.'
- On public enumeration holder block 'LIB_PE:PUBLIC1' because Phase block type 'LIB_PHASE:FUNCTION1' has a parameter named F3 and F5 of enumeration type that are using enumeration sets name #EN1 and #EN2. Enumeration sets #EN1 and #EN2 are defined in public enumeration holder block 'LIB_PE:PUBLIC1.'

The following figure illustrates the export and import of Unit class 'Distribution.'



LIB_MAP:SET1 Block Type [Read-Only]

	Function Name *	Function Description	Library Name *	Phase
1	FUNCTION1	FUNCTION1	LIB_PHASE	PHASE_1
2	FUNCTION2	FUNCTION2	LIB_PHASE	PHASE_2
3	FUNCTION3	FUNCTION3	LIB_PHASE	PHASE_3
4	FUNCTION4	FUNCTION4	LIB_PHASE	PHASE_4
5	FUNCTION5	FUNCTION5	LIB_PHASE	PHASE_4
6	FUNCTION6	FUNCTION6	LIB_PHASE	PHASE_4
7	FUNCTION7	FUNCTION7	LIB_PHASE	PHASE_4
8	FUNCTION8	FUNCTION8	LIB_PHASE	PHASE_4
9	FUNCTION9	FUNCTION9	LIB_PHASE	PHASE_4

LIB_PHASE:PHASE_1 Block Type

	Parameter name*	Parameter description	Data type *	Enumeration Set Name	Size	Default value	Minimum value	Maximum value
1	F1	FORM1	BOOLEAN			☐		
2	F2	FORM2	INT32			0	-2147483648	2147483646
3	F3	FORM3	ENUM	#EN1		zero		
4	F4	FORM4	FLOAT64			NAN	NAN	NAN
5	F5	FORM5	ENUM	#EN2		zero		
6	F6	FORM6	STRING		8	hts1		
7	F7	FORM7	BOOLEAN			☐		
8	F8	FORM8						
9	F9	FORM9						
10	F10	FORM10						
11	F11	FORM10						
12	F12	FORM12						
13	F13	FORM13						
14	F14	FORM14						

LIB_PE:PUBLIC1 Block Type [Read-Only]

	EnumerationSet
1	#EN1
2	#EN2

Figure 32: An example for export and import of a Unit class

14.3 About import/export of Proxy strategies

Proxy strategies can be exported or imported using the Import/Export functionality available in Control Builder. Proxy strategies can include the following:

- cluster representing remote server cluster
- proxy node representing controller configured in the remote server cluster
- proxy tags that represent recipes/procedures on the remote server cluster

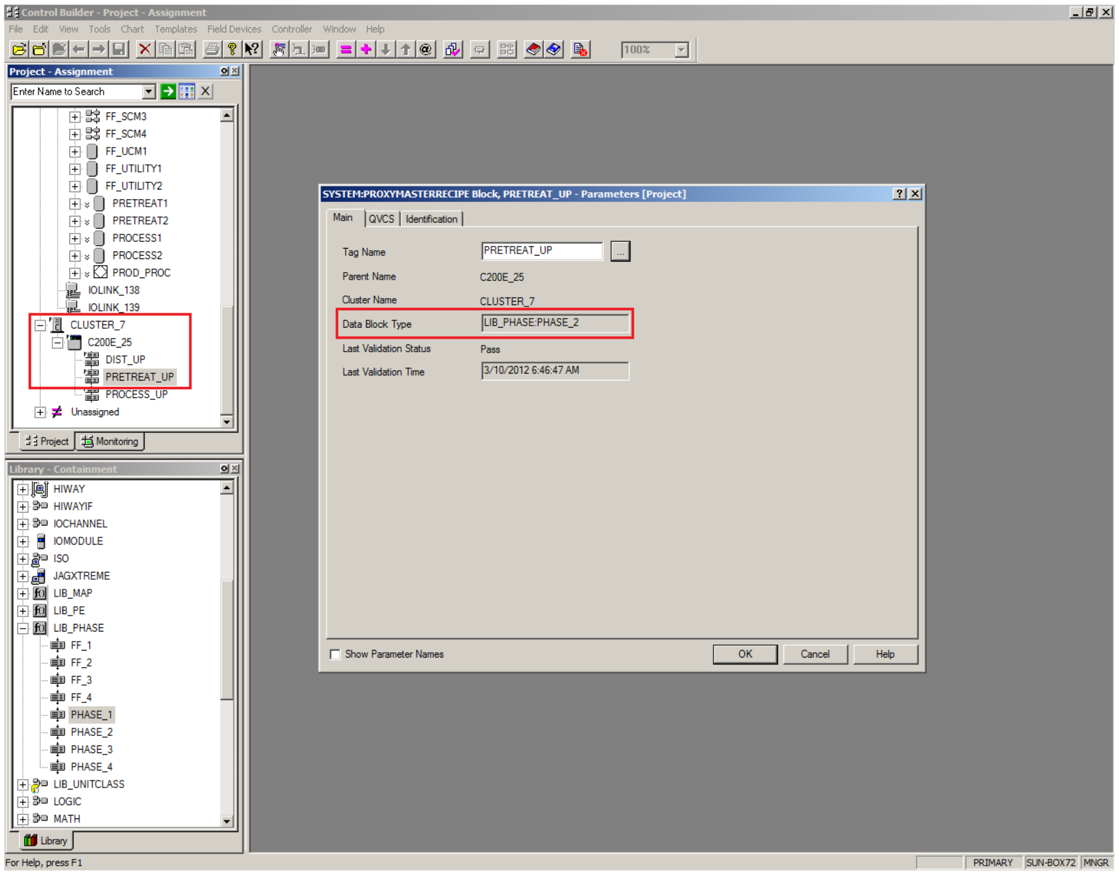
Proxy tags can be one of the three types, that is, proxy Master Recipe, proxy RCM, or proxy SCM. Proxy cluster does not have any dependencies and can be imported or exported independently. Proxy node has a dependency on proxy cluster since it is an assignment parent of proxy node. To import a proxy node, ensure that either the proxy cluster is imported along with the proxy node or it exists before importing the proxy node.

Proxy tags (proxy Master Recipe, proxy RCM, and proxy SCM) have a dependency on proxy nodes since it is an assignment parent of proxy tag. Proxy tags can have a dependency on a Phase block type if they are using it as a Data block. Dependencies of proxy tag can also be multilevel caused indirectly by dependencies of components on which it is dependent. For example, if a proxy tag depends on a Phase block type and if that Phase block type uses Public enumeration holder block then the proxy tag has a dependency on that Public enumeration holder block. To import proxy tags, ensure that either all its dependencies are imported along with it or they already exist before importing the proxy tag.

In the following example proxy tag ‘PRETREAT_UP’ is dependent on the following:

- On Phase block type ‘LIB_PHASE: PHASE_2’ because it is using it as a Data block
- On proxy node ‘C200E_25’ because it is an assignment parent
- On Cluster block ‘CLUSTER_7’ because it is a cluster block and parent of proxy node ‘C200E_25’

The following figure illustrates the export and import of the proxy tag ‘PRETREAT_UP’.



Tag Name	PRETREAT_UP
Parent Name	C200E_25
Cluster Name	CLUSTER_7
Data Block Type	LIB_PHASE:PHASE_2
Last Validation Status	Pass
Last Validation Time	3/10/2012 6:46:47 AM

Figure 33: An example for export and import of a proxy tag

15 Planning for intercluster peer-to-peer

Related topics

"Distribution of recipe layers across clusters" on page 88
"Guidelines for configuring intercluster peer-to-peer communication" on page 89
"Considerations for Phase block type mastership configuration " on page 90
"Considerations for selecting communication ranges for controllers" on page 91

15.1 Distribution of recipe layers across clusters

Guidelines

You can split an Operation and a Phase, or a Procedure and a Unit Procedure across clusters. In other words, Phases may reside in a different cluster from their parent Operations, and Unit Procedures may reside in a different cluster from their parent Procedures. However, Operations must reside in the same cluster as the parent Unit Procedure. This is because unit passing between the parent and child in different clusters is not supported. It is common that the unit needed by the Operation is selected in the Unit Procedure level and the selection is passed to the child Operation. To do so, you must put the Operation level recipes in the same cluster as their parents Unit Procedures.

Restrictions

A child recipe in a peer cluster can be a Class-based recipe or an RCM/SCM. However, there is a restriction to display navigation, if the child in the peer cluster is a Class-based recipe. You will not be able to navigate to the child Control Recipe display from the parent recipe detail display. The navigation always takes you to the detail display of the Master Recipe.

15.2 Guidelines for configuring intercluster peer-to-peer communication

You must adhere to the following guidelines while configuring intercluster peer-to-peer communication.

- If the clusters participating in intercluster peer-to-peer are in a different FTE community, you need to establish communication between the clusters through a router.

  Attention

  For more information about the router configuration, hardware requirement, and network topology for intercluster peer-to-peer communication, refer to the Fault Tolerant Ethernet (FTE) Best Practices document.

- The IP addresses of the clusters participating in intercluster peer-to-peer communication must be configured in the server hosts file.

  For more information about configuring host file, refer to the *Server and Client Configuration Guide*.
- The proxy node name and the proxy tag name must be unique in local and origin clusters.
- Server DSA must be correctly configured on each cluster, which is participating in intercluster peer-to-peer communication, to ensure that the displays function accurately in the Station. All servers in these clusters must be configured to subscribe to both alarm and data. For more information about configuring DSA and subscribing to alarms and data, refer to the "Configuring Distributed System Architecture" section of the *Server and Client Configuration Guide*.
- The IP addresses of the CEEs participating in intercluster peer-to-peer must be unique across clusters.

15.3 Considerations for Phase block type mastership configuration

Phase block type mastership determines the master cluster of a Phase block type and only a master cluster can modify the Phase block type. You can import the Phase block type to another cluster, which has a different cluster ID from the Phase block type master, the destination cluster becomes the slave cluster. You cannot modify the imported Phase block type. For example, consider that the cluster (SERVER1) is auto-assigned with the mastership option and configured as master cluster of a Phase block type (AGIT). In this scenario, when you export the Phase block type (AGIT) and import it to another cluster (SERVER2), the cluster (SERVER2) becomes the slave cluster. You cannot manipulate the Phase block type in the cluster (SERVER2).

For more information on configuring Phase block type mastership configuration, refer to the "Configuring intercluster peer-to-peer" chapter in the *Batch Implementation Guide*.

15.4 Considerations for selecting communication ranges for controllers

Before you configure intercluster peer-to-peer, you need to define the communication ranges for controllers participating in intercluster peer-to-peer. The communication range can be defined from 1000 to 3000. This range is defined in the **Batch Preferences** > **Manage intercluster peer-to-peer** tab. Although, there are no set guidelines for selecting the communication range, you need to consider the following while setting this range:

- Define the range after a clean install or migration. Though default values are set, it is recommended to review and re-configure as necessary.
- Consider future expansions. You may want to add a new controller for participating in intercluster peer-to-peer.
- Controller reload is mandatory in case of a change in communication range. Hence, it is recommended not to change the communication range often.

16 Notices

16.1 Documentation feedback

You can find the most up-to-date documents on the Honeywell Process Solutions support website at:

http://www.honeywellprocess.com/support

If you have comments about Honeywell Process Solutions documentation, send your feedback to:

hpsdocs@honeywell.com

Use this email address to provide feedback, or to report errors and omissions in the documentation. For immediate help with a technical problem, contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the "Support and other contacts" section of this document.

16.2 How to report a security vulnerability

For the purpose of submission, a security vulnerability is defined as a software defect or weakness that can be exploited to reduce the operational or security capabilities of the software.

Honeywell investigates all reports of security vulnerabilities affecting Honeywell products and services.

To report a potential security vulnerability against any Honeywell product, please follow the instructions at:

https://honeywell.com/pages/vulnerabilityreporting.aspx

Submit the requested information to Honeywell using one of the following methods:

- Send an email to security@honeywell.com.

  or

- Contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

16.3 Support

For support, contact your local Honeywell Process Solutions Customer Contact Center (CCC). To find your local CCC visit the website, https://www.honeywellprocess.com/en-US/contact-us/customer-support-contacts/Pages/default.aspx.

16.4 Training classes

Honeywell holds technical training classes on Experion PKS. These classes are taught by experts in the field of process control systems. For more information about these classes, contact your Honeywell representative, or see http://www.automationcollege.com.