Honeywell

Technical Information

Experion LX CAB Specification

LX03-350-120

Release 120

February 2015, Version 1

Revision History

Revision	Date	Description
1	February 2015	Release version

Table of Contents

1. Product Introduction

1.1. Experion LX System

As a member of Honeywell's Experion family, Experion LX is specifically designed to meet the customer needs in mid-tier markets (Chemicals, Industrial Power, F&B, Bio-fuels, …), through integrating state-of-the-art technology from the award-winning Experion Process Knowledge System (PKS) with innovative design of Series 8 I/O modules and cabinets, validated wider range of COTS options, easier engineering and maintenance capabilities, and integrator-friendly programs and tools. Experion LX is the perfect platform for process, asset and business management, and enables customers to increase their profitability and productivity and accessibility to local support without sacrificing quality and reliability in an increasingly competitive environment.

1.2. Architecture Overview

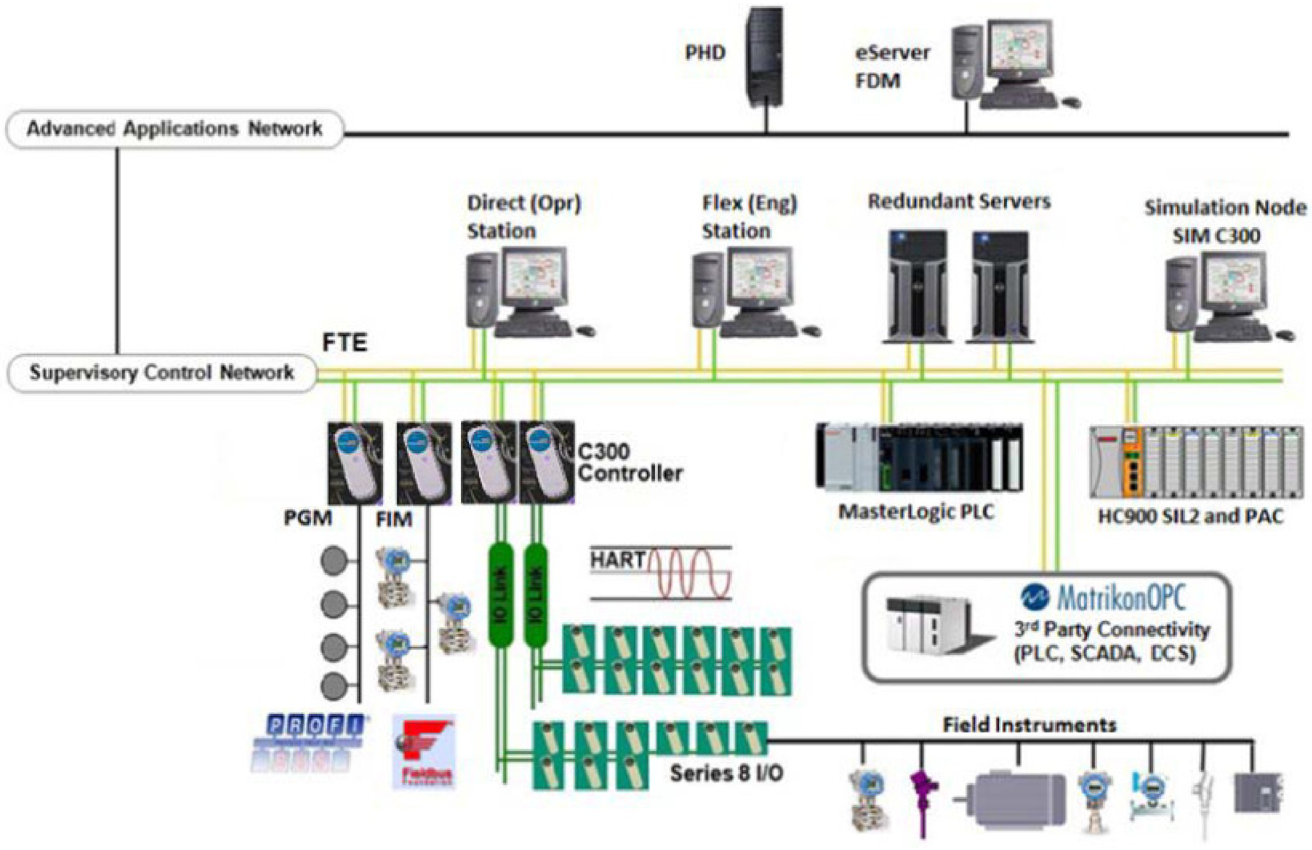

Figure 1. Sample Experion LX architecture

The Experion LX platform comprises many different integrated hardware and software solutions depending upon the needs of the application. This pictured architecture is a representation of many of the possible nodes that can be used in the Experion LX architecture. Note that the architecture is highly scalable and not all nodes are necessary or required.

1.3. C300 Controller Overview

Honeywell’s C300 Controller provides powerful and robust control for the distributed control system (DCS). The C300 is a node in operating Honeywell’s field-proven deterministic Control Execution Environment (CEE) core software. The CEE software provides a superior control execution and scheduling environment. Control strategies for each controller node are configured and loaded through a common Control Builder, an easy and intuitive engineering tool.

1.4. Custom Algorithm Block (CAB)

Honeywell’s Custom Algorithm Block (CAB) allows users to define custom control strategies, offering flexible and tight control using complex calculations running directly in the control environment. These blocks supplement standard Experion LX platform function blocks, and with the capability of running directly in the C300 controller provide the full robustness required for a critical process control application.

2. CAB Overview

2.1. Custom Algorithm Block

A Custom Algorithm Block (CAB) is similar in purpose and structure to the standard function blocks that are distributed with Experion LX Control Builder. Examples of standard function block types include regulatory control blocks, such as PID and RAMPSOAK. Standard block types can be instantiated and used in Control Modules. These blocks have a predefined algorithm and a predefined data structure. By contrast, CABs have user-defined algorithms and data structures.

Once a CAB has been created, it behaves and acts the same as any standard function block. It resides in a Control Builder library and can be used in a Control Module. Its user defined parameters; alarms, algorithm, etc. can be accessed by the Experion LX Server, Stations, and C300 controllers. In addition, a CAB type can be exported from and imported into Control Builder just like a Control Module and used with the C300 CEE. The CAB types appear within Control Builder in user-defined libraries and are dragged into a Control Module to create CAB instances.

A CAB Developer license is required to create and edit CAB Types. The algorithm is programmed in Visual Basic, which is described in more detail in the “CAB/CDB Description and Features” section of this document. A CAB type supports two types of user-defined parameters: Custom Data Parameters (CDPs) and Parameter References (PRefs) that can be externally exposed on the block.

2.2. Custom Data Block

The Custom Data Block (CDB) is a function block that stores data whose type and structure are defined by the user, a Custom Data Parameter (CDP). CDB types are similar to CAB types in that they allow the creation of a custom function block type that can be instantiated multiple times. They differ in that they do not support the definition of parameter references (PRefs), they do not support the definition of an algorithm and they do not require a CAB developer license. Custom Data Blocks are created with Control Builder and can be used in any CEE environment; they can be imported and exported in the same manner as a CAB type.

In all other respects, CDBs are analogous to CABs and standard function blocks. They support associated properties forms and can be designed to support loadable configuration parameters.

2.3. CAB/CDB Description and Features

CAB functionality enables customers to implement solutions to control problems that are not easily supported with standard blocks. CAB types can be used to protect your investment in custom control programs on other control platforms when migrating to Experion LX. CAB provides the support for custom algorithms and custom data. Existing programs can be migrated to Visual Basic, which is a powerful, yet user-friendly programming language.

CAB types are created using Control Builder and the CAB Developer environment. A CAB Developer license is required to activate the CAB development features. The CAB Developer license provides access to the CAB Developer environment, which includes Microsoft Visual Studio development environment and the Parameter Definition Editor (PDE); both are fully integrated in Control Builder.

2.4. Import/Export

Both CAB and CDB types can be exported and imported between Experion LX systems using the standard Control Builder import/export function. Systems that import CAB and CDB types are not required to have a CAB Developer license. Control Builder has the ability to import and create CAB or CDB instances from imported CAB and CDB types.

2.5. Language Constructs

The CAB Developer environment supports Visual Studio 2008's Visual Basic syntax and namespaces and constructs. The following tables list some key language constructs and arithmetic functions that are supported within the CAB Developer environment:

CAB Developer Language Constructs		
Relational expressions (AND, OR, NOR, etc.)	Looping	String manipulation
User defined functions	If-Then-Else	Time manipulation

CAB Developer Standard Math Functions		
Abs()	Exp()	Sin()
Atan()	Int()	Sqrt()
Cos()	Ln()	Tan()

2.6. Supported Variable Types

The CAB Developer environment supports the use of block scope variables, local variables, CAB fixed definition parameters (FDPs), and two types of CAB user defined parameters in developing CAB types and their algorithms. The two types of CAB user defined parameters are custom data parameters (CDPs, also referred to as Value CDPs) and parameter references (PRefs).

2.7. User Application Definition and Control

2.7.1. Algorithm Definition – Execute()

The Execute() subroutine forms the user-supplied interface for all CABs. A CAB Execute() is called whenever the containing CEE is in 'Run' and the parent Control Module is 'Active'**.** The CAB execution is based on the execution period assigned to the control module.

2.7.2. CAB Program Execution Control

The parent Control Module triggers CAB instance execution in the same manner as any other component block within the Control Module. The CAB's order of execution with respect to other component blocks is determined at configuration time. Alternatively, CABs can have their execution controlled by one of the other component blocks within the parent Control Module. This could occur when a CAB serves as an insertion point algorithm. In this case, the partner component block would be configured to call the CAB at an appropriate point in its execution, while the CAB's parent Control Module would be configured to not call the CAB at all.

2.7.3. CAB Program Execution Mode

The execution mode of CAB programs can be characterized as "atomic" or "distributed."

The "atomic" execution mode means that execution is indivisible by requests to access the data it owns. This means that programmers who write CAB programs can do so without concern that data (CDPs) might be read before they are ready, and not while their values are under computation. After the Execute() function initiates, it runs to completion. It is never interrupted by data access requests from outside the block. Atomic execution is supported by the C300. The C300 only supports atomic execution.

The "distributed" execution mode means that CAB blocks can have their execution span several normal execution cycles, and enables additional CAB functionality such as history access and file access. This is only supported in the ACE.

CAB programs are not intended to substitute Sequential Control Modules (SCMs) as sequence programs. Sequence block types or sequence programs are designed with built-in preemption mechanisms that are not directly available to CAB blocks. However, SCMs can work with CAB instances to augment their own function.

2.8. Added Value Built-in Functions

In addition to the rich set of Visual Basic .Net functions, Honeywell has provided a set of additional built-in functions that CAB supports. The additional built-in functions allow for rich integration with the Experion LX system. The following table lists some added built-in functions:

Function	Description
Send()	Subroutine Send() allows text to be sent to the operator message display. Messages appear in those consoles mapped to the area assigned to the parent CM of the CAB instance. Send() supports a single string argument that can be up to 132 characters long. Each string sent corresponds to one line in the message display. The Send() message can be dynamically created using string functions and CAB parameters.
Abort()	Subroutine Abort() causes the program to break the current cycle of execution.
Restart()	Property Restart() allows code executing under the Execute() calling tree to detect whether one of a set of system events has occurred. The set of events that can be detected is defined by the enumeration RestartEnum.

Enhanced Visual Studio Math Functions	
Avg()	Returns the average of the arguments. The function supports the CABMath namespace that includes multi-variable and array support.
Max()	Returns the largest value in a set of values. The function supports the CABMath namespace that includes multi-variable and array support.
Min()	Returns the smallest value in a set of values. The function supports the CABMath namespace that includes multi-variable and array support.
Sum()	Returns the sum its arguments. The function supports the CABMath namespace that includes multi-variable and array support.

2.9. Protected Environment

CAB is a fully protected programming environment that protects the user against catastrophic mistakes like terminating the whole controller environment due to writing to wrong memory locations, or pausing execution of the controller environment while waiting for operator input.

The CAB development environment provides standard exception handling and alarms, with configurable priority, for block terminations, PRef read/write errors, and code exceptions. CAB blocks terminate on memory violations and if excessive execution is detected. .

2.9.1. Function Limiter & Supported Namespaces

The function limiter in CAB catches code that is not allowed and displays an error message in the output window within the CAB Developer environment. CAB does not support declare statements that allow you to access functions in external DLLs. The tables below show the namespaces and classes allowed for CAB/C300.

Allowed for CAB/C300 applications			
Functional Area	**.NET Namespace**	**Class / Structure**	**Method**
.NET run-time Compiler Services.	System.Runtime.CompilerServices.RuntimeHelper	Any	Any
Fundamental .NET Classes and Structures	System	Activator, Array, BitConverter, Buffer, CharEnumerator, Convert, DBNull, Environment[1], EventArgs, Math, Object, OperatingSystem, Random, ResolveEventArgs, String, TimeZone, Type, ValueType, Version ArgIterator, Boolean, Char, DateTime, Double, Int32, Int64, TimeSpan, UInt32, UInt64	Any

Allowed for CAB/C300 applications			
Functional Area	**.NET Namespace**	**Class / Structure**	**Method**
Various Collections Classes	System.Collections	Any	Any
Various Specialized Collections Classes	System.Collections.Specialized	Any	Any
Type converters	System.ComponentModel	Any Type Converter class	Any
Culture-related information	System.Globalization	Any	Any
Text and String manipulate classes	System.Text	Any	Any
.NET regular expression engine	System.Text.RegularExpressions	Any	Any
Note 1 – Setting the property Environment.CurrentDirectory is not allowed.			

2.10. CAB Debugging

There are several methods available to debug and test CAB applications. The most basic debugging method is using the SEND() function, which sends messages to the Experion LX system message summary. The combination of SEND(), custom alarming, and outputs to the custom defined parameters affords programmers with a means of straightforward debugging.

3. CAB Specifications and Sizing

3.1. Key Features of CAB/C300

- Execution Method – CAB/C300 supports only periodic, atomic execution in which the algorithm runs from start to finish during a short interval of time that recurs every CM period.
- Execution Base Cycle – The base execution cycle for a C300 CM is 50ms. This allows for execution from once every 50ms to once every 2 seconds.
- File / History / Database Access – The C300 has no local disk. So it does not support File / History / Database Access.
- Fixed Parameter References – CAB/C300 supports fixed parameter references
- Multidimensional arrays –CAB/C300 supports arrays with up to three dimensions.

3.2. Processing and Memory Usage

The amount of CPU consumed by the execution of a CAB/C300 block instance can vary widely depending on the line count, the amount of looping and the program's general complexity. As one example, the following table gives an indication of the CPU consumption for one instance of the "Medium" CAB type.

CPU Utilization of Medium CAB Type				
CAB Size Category	**Lines of Code**	**Number of CDPs**	**Number of PRefs**	**PU / Block Instance**[1]
Medium	100	15	50	6.3
Note 1 – In CEE one Processing Unit (PU) is equal to ¼ the total amount of time required to process a typical DevCtl CM and a typical RegCtl PM.				

CAB/C300 types and instances each use CEE memory. The first time an instance of a CAB/C300 type is loaded to CEE, the block type is loaded with it. The first instance thus requires a total amount of memory equal to that used by the type and the instance together. When the second and subsequent instances of a type are loaded, the only additional memory required is that used by the instance.

The table below shows type and instance memory used by three kinds of block type, characterized here as "Small", "Medium" and "Large". These characterizations are approximate. Types and instances substantially larger than what is called "Large" in the table below can be loaded to a C300.

Memory Use of Sample Types and Instances					
CAB Size Category	**Lines of Code**	**Number of CDPs**	**Number of PRefs**	**Block Type Size (KB)**	**Block Instance Size (KB)**
Small	20	1	10	21.6	1.5
Medium	100	15	50	32.8	4.7
Large	200	30	100	45.9	8.6

CAB/C300 differs from CAB in that the memory used by both block type and block instance come exclusively from CEE's user memory pool. There is no additional memory drawn from a separate pool managed by the C300 OS.

3.3. CAB Type and Instance Quantities

The table below indicates the number of CAB types and instances that a C300 controller supports. The maximum numbers depend on available memory and processing units. The column "typical count" indicates the possible numbers of CABs and instances.

CAB Types and Instance Specifications			
Item	**Typical Count**	**Maximum Count**	**Comment**
Loaded CAB Types	30 to 70	100	This limit is enforced and adds to the constraint of memory utilization within the CEE user memory pool.
CAB Instances Per Type	1 to 10	No enforced limit	The only limit on instances per type is memory utilization within the CEE user memory pool.
Total CAB Instances Across All Types	50 to 500	No enforced limit	The only limit on total instance count is memory utilization within the CEE user memory pool.

3.4. CDB Processing and Memory Usage

The amount of PUs consumed by a CDB instance is zero. The memory usage depends upon the size and complexity of the CDB type.

PU and MU Usage for a Honeywell-defined Medium CDB

CDB Size Category	Parameter Defined	PUs	MUs

Medium	CDB instance has a total of 15 Custom Data Parameters of which 2 are arrays:
	5 32 char. STRING;
	4 scalar FLOAT64;
	4 scalar BOOLEAN;
	1 20 element FLOAT64 array;
	1 20 element BOOLEAN array.
		0	1.5 (for first module instance)
			
			1.0 (for every subsequent module instance)

Note: Applies for C300.			

3.5. CDB Type and Instance Quantities

The table below indicates the number of CDB types and instances that a C300 node supports. The maximum number depends on the available C300 user memory and processing units.

CDB Types and Instance Specifications		
Item	**Max Count**	**Comment**
Loaded Instantiated CDB Types	400[1]	CDB infrastructure prevents the number of loaded *instantiated* [2] block types per C300 CEE from growing beyond the indicated maximum number. There are no enforced limits for how many CDB types that can be stored in the ERDB (Engineering Repository Database).
CDB Instances Per Type	--	The number of supported CDB instances is constrained by the available C300 user memory pool.
Total CDB Instances Across All Types	2000	These are recommended limits. The number of supported CDB instances is constrained by the available C300 user memory pool.
Note 1: This limit is independent from the CAB or standard Function Block type limit. Note 2: An instantiated CDB type is one that has one or more instances defined against it.		

3.6. CAB Developer Specifications

Multi-user CAB Developer Limits	
Maximum concurrent CAB Developer Clients connected to single	12

Server [1]	
Note 1: The number of CAB Developer Licenses (maximum of twelve per system) needs to be lower or equal to the number of CB Client licenses CV-COBLDR of which one is delivered with the Experion LX Base software.	

3.7. CAB/CDB Parameter Specifications

CAB and CDB blocks support fixed and user defined parameters. Fixed Definition Parameters (FDPs) are Honeywell defined parameters that are default on all CAB and CDB block types. User Defined parameters are made up of two types, Custom Data Parameters (CDP or Value CDP) and Parameter References (PRefs).

CDPs are supported by both CAB and CDB and have the following characteristics:

- Stores data local to the CAB block
- Parameters are exposed system wide
- Used similarly as a local variable in a Visual Basic program, only values are persistent from one execution cycle to the next.
- Supports up to 2 dimensional array definition

PRefs provide a form of indirection and are supported by CAB (but NOT by CDB):

- Stores data external to the CAB block
- Is an alias to a parameter (e.g., Flow.Val = FI101.AI.PV)
- Parameter exposed system wide
- Similar to a pointer in C programming

There is a limit to how many parameters a CAB can have. CDPs, FDPs, and parameter references all contribute to a total that is constrained.

Configuration Limits for CAB Types and Instances		
Parameter Type	**Max Limit**	**Comment**
FDP (Fixed Definition Parameters)	200	FDPs are defined by Honeywell and are not available for definition by users although users can read them and write to some of them. Approximately 50 are currently used and 150 are reserved for future use.
User Defined Parameters	C300 - 1500[1]	Custom Defined Parameters (CDPs) count as **one** user defined parameter, Reference parameters (PRefs) count as **two** user defined parameters
Note 1: Array parameters count as one parameter no matter how many elements they have.		

3.8. Data Types for CDPs and Parameter References

CDPs can be indexed parameters (logical arrays) of up to two (2) dimensions. When configuring array parameter definitions within the Parameter Definition Editor window, the data type is selected from one of the scalar types listed below. The CDP index is specified by parameter attributes distinct from the data type.

Parameter references cannot be indexed (logical arrays of references are not supported). However, they can be used to refer to scalar elements of array parameters at fixed index.

Data types supported for CDPs within CABs and CDBs, and for parameter references within CABs, are shown in the following table. Note that the set of types supported for data that exists within the Experion LX name space is more restricted than the set supported within the confines of a CAB program. Within a CAB program, the only limitations are the capabilities of VB.NET.

Data Types for CDPs and Parameter References

Data Type	Supported for CDPs	Supported for Parameter References	Comment
BOOLEAN	Yes	Yes	
INT16	Yes	Yes	The CDP or PRef is defined as a type INT32 or FLOAT64. Automatic conversion takes place from the referenced type to the CDP or PRef type.
INT32	Yes	Yes	For CAB parameter references, implicit conversion operations are supported between INT32 and most other scalar types.
<Standard System Enumeration>	No	No	Within CAB programs, one can use the equivalent numerical values when dealing with system enumerations. The user can also define program-local enumerations that match both the system enumerations and the self-defining enumerations. When storing to external parameters, CAB parameter references support implicit conversion so that the store will be accepted by the destination parameter.
<Self-Defining Enumeration>	No	No	
FLOAT32	Yes	Yes	The CDP or PRef is defined as a type FLOAT64. Automatic conversion takes place from the referenced type to the CDP or PRef type.
FLOAT64	Yes	Yes	Within CAB programs, FLOAT64 CDPs and PRefs are of the VB.NET data type Double.

Data Types for CDPs and Parameter References			
Data Type	**Supported for CDPs**	**Supported for Parameter References**	**Comment**
TIME	Yes	Yes	Parameters of type TIME indicate a point in time (social time). Within CAB programs, TIME is represented by the .NET structure DateTime.
DELTATIME	Yes	Yes	Parameters of type DELTATIME indicate a time span (time duration). Internal to CAB programs, DELTATIME is represented by the VB.NET structure called TimeSpan.
TIMEOFDAY	Yes	Yes	Parameters of type TIMEOFDAY (Time Of Day) indicate the time elapsed since midnight. CDPs of type TIMEOFDAY are supported by both CABs and CDBs
STRING	Yes	Yes	String types can be up to 255 characters in length.
Structure	No	No	Structure types can be used internal to CAB programs, but CDPs and parameter references cannot have structure type.
Class	No	No	Class types can be used internal to CAB programs but CDPs and parameter references cannot have class type.

In the table above, it is important to understand that typing of CAB CDPs and CAB parameter references is described in terms of the Experion LX system type names. This is necessary as CDPs are part of the Experion LX namespace and parameter references point to parameters that are part of that namespace. However, there is a one to one mapping between each of the supported data types listed above and the VB.NET data type used within a program. This is shown in the following table:

Data Type mapping between Experion LX and VB.NET	
Data Type Of Parameter As Viewed Externally In Experion LX Namespace	**Data Type Of Alias As Viewed Internally Within VB.NET Program**

Data Type mapping between Experion LX and VB.NET	
Data Type Of Parameter As Viewed Externally In Experion LX Namespace	**Data Type Of Alias As Viewed Internally Within VB.NET Program**
BOOLEAN	Boolean
INT32	Int32
FLOAT64	Double
TIME	DateTime
DELTATIME	TimeSpan
TIMEOFDAY	TimeSpan
STRING	String

4. Glossary

Term or Acronym	Description
C300	A specific type of Honeywell Process Controller based on the series 8 form factor
CAB	Custom Algorithm Block
CDA	Control Data Access is the Experion LX system communication infrastructure and data access interface schema that provides application integration with Experion LX system objects.
CDB	Custom Data Block
ControlNet	Real-time control-layer network.
Experion LX Engineering Station	The node (optionally redundant) at the heart of Experion LX. The Engineering Station encompasses a wide range of subsystems including history collection, SCADA interfaces, alarm/event, etc.
LCN	Local Control Network
OPC	Series of standard specification for open connectivity in industrial automation, originally based on Microsoft's OLE COM and DCOM technologies.
PPS	Parameters per second

Experion® is a registered trademark of Honeywell International Inc.

All other products and brand names shown are trademarks of their respective owners.

This document contains Honeywell proprietary information. It is published for the sole usage of Honeywell Process Solutions' customers and prospective customers worldwide. Information contained herein is to be used solely for the purpose submitted, and no part of this document or its contents shall be reproduced, published, or disclosed to a third party without the express permission of Honeywell International Inc.

While this information is presented in good faith and believed to be accurate, Honeywell disclaims the implied warranties of merchantability and fitness for a particular purpose and makes no express warranties except as may be stated in its written agreement with and for its customer.

In no event is Honeywell liable to anyone for any indirect, special or consequential damages. The information and specifications in this document are subject to change without notice.

For more information
To learn more about Honeywell's products or solutions visit our website www.honeywellprocess.com or contact your Honeywell account manager.

Automation & Control Solutions
Process Solutions
Honeywell

1250 West Sam Houston Parkway South
Houston, TX 77042

Honeywell House, Arlington Business Park,
Bracknell, Berkshire, England RG12 1EB UK

Shanghai City Centre, 100 Junyi Road
Shanghai, China 20051

www.honeywellprocess.com

LX03-350-120
February 2015
© 2015 Honeywell International Inc.

Honeywell