MilramX Architecture

Introduction

MilramX is a set of software that is used in conjunction with Microsoft's Visual Studio .Net IDE (Integrated Development Environment) to provide over 90% of the code needed to implement applications that:

1. Need to periodically examine data in a variety of systems and databases for changes.
2. Need to process that data and pass it on to other systems or to use it to provide business information for managers, employers, customers and suppliers.
3. Can run autonomously 24x7 without intervention by IT staff but which can automatically alert an IT person when human interaction is required.
4. Can handle a wide variety of data formats, including knowing how to process bad or incomplete data with minimal human intervention.

The MilramX software development framework (BC) is used as the basis for:

1. The BellHawk SCI Supply Chain Integrator where it is used to implement automated data exchange interfaces between different systems. Here it is primarily used to implement interfaces between the BellHawk barcode tracking software and accounting and ERP systems as well as with logistics and EDI systems.
2. BellHawk RTX Real-Time Intelligent Operations Monitoring and Alerting software where it is used to implement systems that periodically examine data in different systems and generate alerts about operational events that need human attention.
3. The BellHawk IOT Internet-of-Things data collector, where it is used to collect data from devices such as RFID portals and to integrate the data they collect into the BellHawk inventory and work-in-process tracking database.
4. The BellHawk Business Intelligence data extractor, where it is used to extract data from BellHawk and other systems and to:
    a. Export this data to "Big Data" databases in such a form as to be useful for subsequent after-action reporting and analysis.
    b. Generate and automatically Email after-action reports to specified managers, customers, and suppliers.
    c. Display performance data on "shop floor" displays.
    d. Create web-portals for customers and suppliers to see limited sub-sets of the data available in the operations tracking systems.

The characteristic of all these applications is that they all need some custom code development but over 90% of the needed code can be provided as pre-developed DLLs (Dynamic Link Libraries) or as automatically generated code. The executable code also needs to be embedded in a framework where it can be run periodically at specified times in a controlled environment where it can be monitored and also generate alerts for IT people when human intervention is required.

This document describes how MilramX provides these capabilities thus enabling this class of application to be implemented rapidly and to be supported and maintained over many years of reliable operation by an IT department.

MilramX Architecture Philosophy

MilramX is deliberately Microsoft "centric" as it is intended to be used in small to mid-sized industrial organizations which, in our experience, predominantly use Microsoft servers and work-stations. It is intended to be installed and run on Microsoft operating system based computers. It can, however, exchange data with a wide variety of computers based on other operating systems and databases.

Applications built with MilramX are typically used within an organization's own private network but can be run "in the Cloud" at remote data centers, if needed. What typically "anchors" MilramX in the plant, however, is the need to interact with legacy systems and equipment that can only be accessed over the local area network.

MilramX is designed for use by, and to be affordable by, small to mid-sized industrial organizations, which have limited software development resources. It is designed to be used by IT staff and manufacturing engineers who can develop code in Visual Basic (VB.Net or C#.Net, if preferred) but who are not expert programmers.

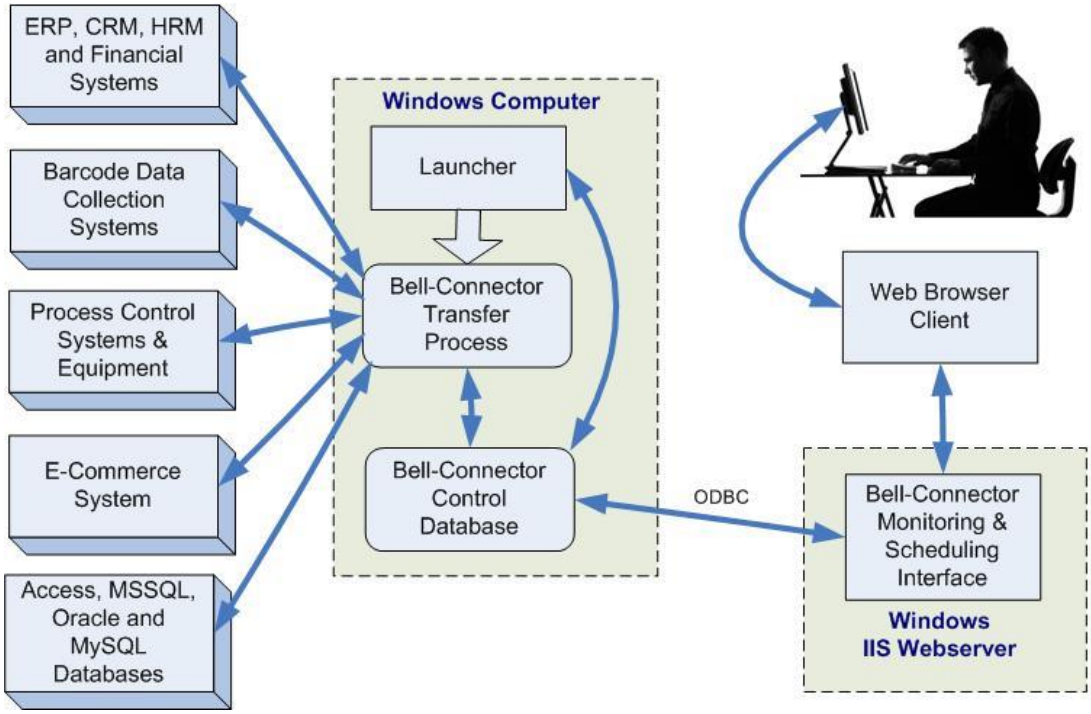
One goal of MilramX to hide all the complexity of tasks such as interacting with different databases, converting data formats, communicating with remote systems over networks, and sending Emails or text messages from the programmers. This enables the programmers to focus on business application specific tasks such as converting data from one high-level business object to another or interpreting and responding to events from different systems.

MilramX also provides an architectural framework, which has been honed over many years of practical application, so that programmers do not have to start from scratch. They can simply use the framework and add that code that is custom to their specific application. Even where the MilramX application code is provided by BellHawk Systems, this is made available as VB.Net code to clients so that they can modify the data exchange or interpretation code, if needed, without requiring BellHawk Systems staff to make the changes.

The MilramX software framework is provided as supported and licensed code by BellHawk Systems. This means that over 90% of the code in any interface, alerting or business intelligence system is well-debugged standard code used by everyone. That leaves less than 10% application specific code to be debugged in each application and this can be developed under the control of each client's own programmers.

This approach also enables programs, such as the BellHawk tracking software, to be run as commercial "off the shelf" software (COTS) without needing customization.

Overall Architecture



MilramX consists of a number of elements:

1. One or more MilramX transfer processes (.exe executable programs) whose job it is to retrieve information from one or more systems, process that information, and take appropriate action. .
2. A MilramX Control Database – this is a SQL Server database is where MilramX keeps all the information that it needs to monitor and control the running of the transfer functions.
3. A MilramX Control Website that enables remote monitoring and control of the transfer processes. Through this interface, users can schedule and monitor the transfers, see and investigate errors, and edit and retry any transfers that failed.
4. A MilramX Launcher program. This Launcher.exe program runs continuously as a background user process. It monitors the setup and status data in the Control database to decide when to run each transfer process and with which parameters. It also tracks for transfer processes that have exceeded their maximum run-time and kills them if needed.

The launcher and transfer-function programs run on a Windows Workstation or Server, typically within a Windows user environment.

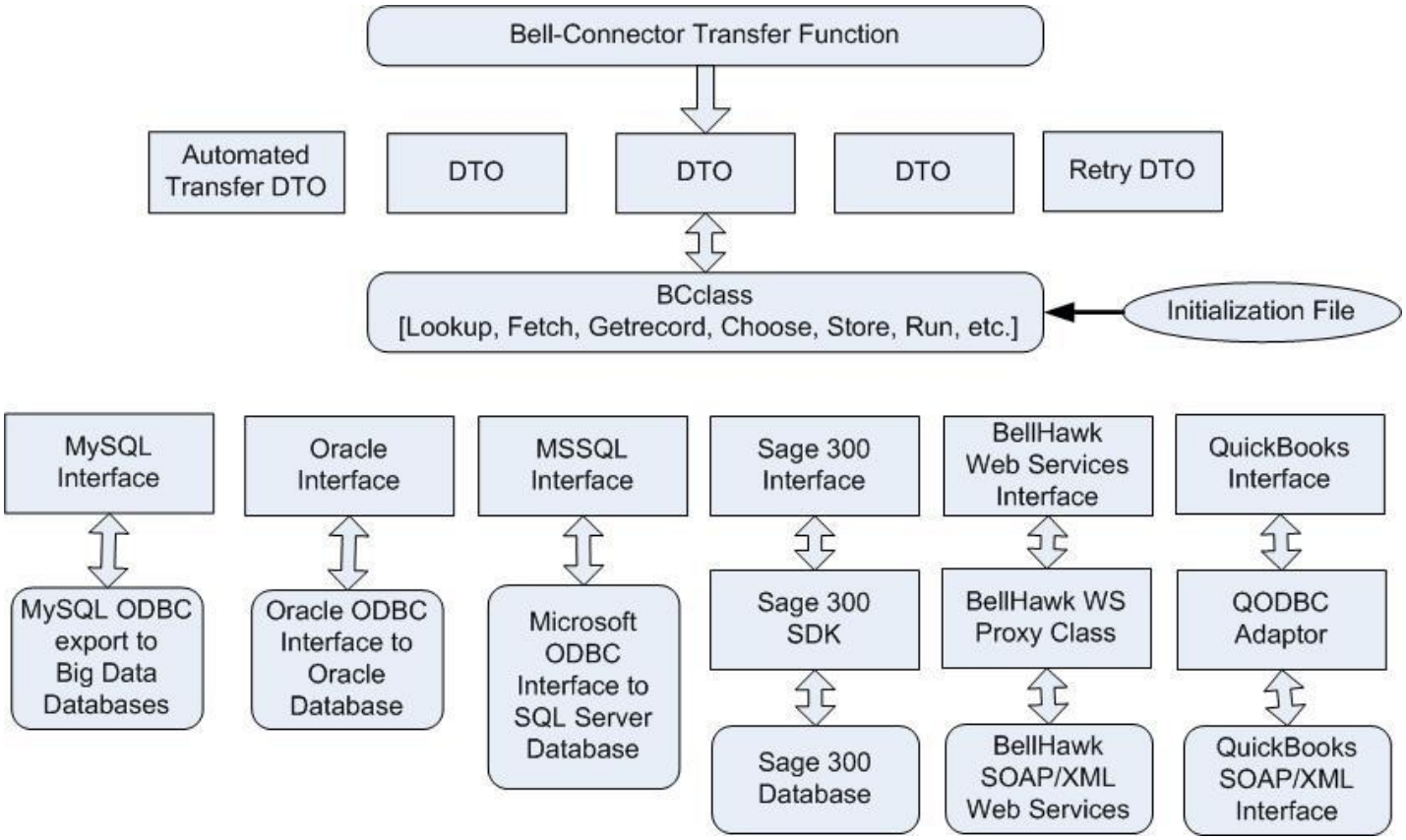
The MilramX Website runs under the control of the IIS webserver. It can be installed on the same Windows computer as the transfer function or it can be run on a separate Windows Server computer, provided that the Windows Server is on the same LAN or VLAN network as the Windows computer used to host the Control database.

MilramX keeps the last-transferred times separate for each transfer function by tagging these times with the name of the transfer function in the Control database. This enables multiple transfer functions to be active at the same time.

The use of the MilramX website to control and monitor the execution of BellHawk is described in the “MilramX User Manual”.

Structure of the Transfer Function

While the MilramX (BC) Launcher will work with any executable program, normally BC transfer functions are developed using the BCclass library and BC interface class libraries, which are supplied as dynamic linked libraries (DLLs).



MilramX transfer functions are normally written as a set of DTOclass objects. These Data Transfer Objects (DTOs) are effectively subroutines that perform specific application functions within the transfer function.

Thus we might have a transfer function that is the interface between an ERP system and the BellHawk operations tracking system. Within this transfer function we might have separate DTOs that are responsible for transferring item master records, purchase orders, and sales orders.

These DTOs do not directly interact with the databases in the different systems. Instead they call BCclass methods, such as Lookup, Fetch, Getrecord, Store and Run to interact with the different systems. These, in turn, call BC interface classes, which handle the actual physical interface to the systems.

The BC interface classes may interact directly with the databases, through the use of ODBC interfaces (which are typically provided by the database developer) or they may interact with the applications themselves, such as through SDKs (software developer kits), web services interfaces, or other third party software.

While it is possible for knowledgeable developers to implement their own BC interface classes, BellHawk Systems typically supplies the BC interface classes pre-built. This then insulates VB.Net developers from having to know about all the (usually very messy) details of interfacing with each system or database.

It is the function of the BC interface class objects to translate between High Level Business Objects (HLBOs) and all the complexity that goes into translating between the lookup or storing of these HLBOs and the reading, inserting or updating of multiple tables, sometimes in multiple databases.

In MilramX (and BellHawk) terms, each HLBO has a unique Keyword that identifies it (such as "PO" for purchase order header or "Item" for an item master record) and a set of name-value pairs of strings, such as "PONumber":"PO9786" and "ItemNumber":"ABC1234".

The VB.Net (or C#) programmer that is writing a DTO performs their application programming in terms of these HLBOs, which are BCclass objects, thereby isolating the programmer from all the complexity of the interfaces. This also enables easy maintenance of the transfer functions, as data exchange requirements change over time. It also isolates the business logic in the DTOs from updates to the ERP or accounting systems or databases with which it is interacting.

The BCclass comes with some pre-written DTOs, such as for providing automated data transfer between like tables in different databases and retrying failed transfers. It also contains many supporting functions for manipulating HLBOs as well as supporting capabilities such automated detection and translation of bad characters and conversion to and from JSON strings.

By convention, we name the transfer function for interfaces something like BHS3I.exe, where the first two letters represent one system (in this case BellHawk) and the second two letters represent the other system (in this case Sage 300) and the I stands for interface. This makes these processes instantly recognizable when using the Microsoft task manager.

When DTO class objects are defined in the code, they are given a name, such as S2BPO (for Sage to BellHawk transfer of PO Header objects). When the transfer function is called, it is called with the index of the DTO entry in the BC_CTL table in the MilramX Control database. This entry contains the name of the DTO class instance to be run, when the transfer function is executed.

This dynamic linking to the named DTO objects enables new DTOs to be added to the transfer function by simply adding them to the transfer function Visual Studio project, without the need to create code to specifically call them. These DTOs can then be setup using the BC Website (see "MilramX User Manual" in the User Manuals section of www.BellHawk.com for details) and are run by the Launcher. The Launcher runs the transfer function with the index into BC_CTL for the named DTO, which is how it knows which DTO to execute.

This enables developers to simply write the DTO code for their specific application without needing to code all the linkages involved.

Adaptors and the Initialization File

The BC interface classes are generic. They are tied to specific databases or URLs through data included in an initialization (.ini) file. We refer to the combination of the specific database or URL and the BC interface as an Adaptor, which is given a name, such as “BellHawk” or “S300”. An interface, such as the MSSQL database can be used by multiple adaptors.

An example initialization file is shown here

```
[Control]
XMLfile = Control.XML
Type = MSSQL
; Replace the values on the next 4 lines with your BellHawk BHCTL SQL server database/login
settings
Server = PGWIN7\SQLSERVER2008          (1)
Database = BellConnectorS300Test
UserID = ********
Password = *********
LogFolderName = F:\DEV\Dev2015\BHS3I\BCWebsite\BCWebsite\Logfiles          (5)
DebugLevel = 0

[Adaptors]
; Adaptors should be a comma-separated list of adaptors          (2)
Adaptors = S300,BellHawk, Control

[BellHawk]
Type = MSSQL
XMLFile = BHMeta_v6.60.XML
Server = PGWIN7\SQLSERVER2008          (3)
Database = V66S300Test
UserID = *********
Password = *********

[S300]
Type = MSSQL
XMLFile = S300.XML
Server = SAGESERVER
Database = SAMINC          (4)
UserID = *******
Password = ********
```

Here we see a definition (1) for the [Control] Adaptor that enables BCclass to communicate with the MilramX control database. We also have a list of the [Adaptors] that the transfer function can access (2) and definitions for the BellHawk (3) and S300 adaptors (4).

This information is made available to the BC interface class objects by the BCclass so that the interfaces, in this case, can form connection strings to connect to the individual databases. Note that the same BC interface class, in this case MSSQL, is used to access different databases, each with its own adaptor name.

Within each adaptor definition is a reference to an XML metadata file that contains the information about the HLDO’s that can be accessed through the BCclass interface using the specified Adaptor. Note that the [Control] adaptor contains definition of objects that can be accessed within the control database itself.

MilramX has extensive error detection and reporting capabilities. As part of this, the BCclass creates a daily log file in a folder (5) specified in the initialization file. The detail level of reporting (informational, warning, error) can be set by the DebugLevel.

Note that the DLL used for the interface is looked up by name by the BCclass based on the Type field. Thus, in this case Type = MSSQL will cause BCclass to reference the routines in MSSQL.DLL.

Please note that the XML Metadata entries contain the name of the adaptor they belong to. If the Adaptor section name is different from that in the XML Metadata then an Alias entry must be used in the initialization file.

For details about setting up the initialization file please see the "MilramX Installation and Configuration Guide" in the User Manuals section of www.BellHawk.com.

For details about how to create and edit XML Metadata files, please see the "WebDexel User's Manual" in the User Manuals section of www.BellHawk.com.

MilramX Database

The MilramX database is the link between the MilramX Website User Interface and the Launcher and the BC Transfer Function.

The MilramX Control database (named BellConnector by convention) contains a number of tables that it is useful to know about:

- BC_CTL – this contains one entry per DTO with data about the name of the executable process and the DTO and parameters such as the Adaptors it uses. These entries are placed here and maintained through the MilramX website interface. Note that the same DTO application code can be used for different transfers by specifying different adaptor names.

- tblTransferDef – each DTO entry can have up to 8 DTO specific parameters in addition to the standard parameters such as period and start-stop time. By convention, for data transfers, the first two parameters are used for the names of the Adaptors to be used by the DTO. This table specifies what parameters will be requested for each DTO. This table is setup through an Excel import using the Sys Admin DEXEL screen in the BC website.

- tblTransferQueue – some BC Interfaces (such as MSSQL) support the use of a queued store. Instead of directly attempting to insert or update an entry in a target database, the transfer entry (in the form of a HLBO) is first written into the Transfer Queue. Then if the transfer is successful, the transfer request may be deleted or it may be left in this table for logging and diagnostic purposes (depending on a DTO setting through the BC website). If the transfer fails, the entry is left in the Transfer Queue where it may be viewed, edited, and retried through the BC website interface.

- tblErrorLog – in addition to writing errors and warnings in the daily log file, the BCclass error handling routines write the errors into the Error Log so that the error history can be viewed through the BC website interface.

- tblUsers – this is where the names and encrypted passwords of users, setup by the BC website Sys Admin, through the BC website Admin switchboard, are stored.

- tblParameters – this is where the log file folder name is stored along with the Email address of the designated Sys Admin and the SMTP transmission parameters, so that the Sys Admin can be notified by an Email alert if an error occurs in a transfer.

- tblParameterStore – this is where MilramX stores the last access times for fetching data for each HLBO keyword through each adaptor. It is used to enable DTOs to get the latest updates to database tables without writing the code to find which entries have changed.

- tblFileSend – this is where DTOs can place entries (using the Control Adaptor) for files to be sent by FTP to an FTP site. In future this will be expanded to include sending files as Email attachments. They actual sending of the files is done by a FileSend DTO which is provided as part of the BCclass DLL and can be scheduled to run periodically to deliver the files.

- tblChars – one of the big problems in collecting data from legacy systems is that they can contain a lot of spurious characters that are unacceptable to the target system. Also the source system may have text data in Unicode or UTF8 format whereas a target system (such as BellHawk) may only accept extended ASCII characters. BCclass, in conjunction with the BC Interface, can perform automated character translation based on the contents of this table, which can be setup using an Excel import through the BC website DEXEL functionality.

- tblMap – this table specifies the one-to-one mapping between a parameter in one HLBO and a parameter in another HLBO. It is setup through a DEXEL import of an Excel spreadsheet using the BC website. This mapping can be used to simplify DTO code development by specifying all the one-to-one mappings in this file and then simply calling a BCclass function within a DTO to have all the one-to-one transfers take place automatically without writing code for each mapping.

- tblAutoTransfer – this builds upon the entries in tblMap to create pseudo DTOs (that is DTOs in name only) in which all the mappings are specified in tblMap. The actual transfers are performed by the AutoTransfer DTO which is provided as part of the BCclass DLL. This file is setup using an Excel file import through the BC website DEXEL screen.

For details about how the formats of the Excel import files, described above, please see the MilramX User Manual.

How a DTO Gets Created and Run

In a Visual Studio development environment, the process is as follows:

1. If this is a new project, start with a project template provide by BellHawk Systems (essentially a main program and a template DTO) and rename the project and its components to match your naming scheme.

2. Write a DTO class object (essentially a VB.Net subroutine) as described in the “MilramX Programmers Guide” as part of your Visual Studio project.

3. Setup the transfer function and DTO name and parameters (adaptors etc.) in tblTransferDef using an Excel import through the BC website DEXEL function, as described in the “MilramX Installation and Configuration Guide”.

4. Use the BC website to setup an instance of the DTO, selecting the adaptors to be used, as well as any additional parameters, and mark it active.
5. Setup the .ini file in the same folder as your main program. Also include the XML metadata files for the systems you will access.
6. Look at the BC_CTL table to get the index of the DTO entry and place this in the call parameters for executing your program in debug mode in Visual Studio.
7. Set a breakpoint at the beginning of your DTO code and run your project in Debug mode and you are on your way.

When ready for deployment, create the .exe file (or an .msi install file) using Visual Studio. Place the .exe file plus the ini file and the metadata files into a folder whose location is specified in the Launcher's own initialization file (see MilramX Installation and Configuration Guide). Then run the Launcher in a user process and it will automatically run the DTOs in sequence as specified in the BC_CTL table by settings made through the BC website.

The Launcher

The launcher is supplied as an executable Launcher.exe program. When run the Launcher uses setup data in its own initialization file to access the MilramX control database through which it communicates with the BC website.

The launcher runs in a loop and every few seconds checks the BC_CTL file for the next DTO to run. When the time arrives, the Launcher then launches the specified transfer function with the index of the DTO as its argument. It then monitors the running DTO process to see when it stops running. The launcher then finds the next DTO and runs it.

The launcher monitors the executable process and can issue a kill request to the operating system for the transfer function process if it runs beyond the execution time specified for the DTO.

The Launcher logs the start time for the DTO and any errors in launching the DTO, in the DTO record's error field. It also sets the DTO's launched status in the DTO record in BC_CTL and logs any errors it encounters in the error log and the daily log file.

It only changes the status to an error status if it has to kill the executing process, when it also sets the end run time for the DTO.

Please note that the launcher is provided as an executable process that is intended to run as a user process. This process can be run as a disconnected user process on a Windows Workstation or a Windows Server computer. The reason for this is that this enables the transfer function to run with all the user's privileges to access databases, files and networks. This makes for very easy setup but the user process does have to be restarted manually whenever the computer is rebooted.

The reason that we do not provide the Launcher as an installable service is that, by default, the transfer function it launches would then be run without the right to access files, databases or networks, thereby defeating the purpose of MilramX.

We have run the Launcher as a service but the setup is difficult and very situation dependent, especially in an Active Directory environment. For those developers who are expert in the arcane

art of setting privileges for processes launched from Services, we do make the launcher available, as is, with no guarantees, in VB.Net source code form, so they can modify it to run as a Service within the Windows operating system.

Note that it is the responsibility of the BCclass to update the DTO entry in BC_CTL when a DTO changes status to start running and then finish running and also when an error is reported.

Error Detection and Reporting

Both the Launcher and the Transfer functions are console applications which run with their console windows hidden unless there is an error.

The BCclass includes an error reporting function BC_Error( ). This is called with an argument indicating whether this is an Error, Warning or informational message and this is followed by a message string. This same error routine is available within the DTO as Me.Msg( ).

How these messages are treated depend on the value of DEBUG in the [Control] section of the initialization file. The places where messages can be logged include:

1. Daily log file: All Error messages, Warnings if DEBUG > 0 or less than -1
   Informational messages if DEBUG> 1 or less than -2
2. Error Log: Error messages, Warnings if DEBUG = -1 or +1,
3. Email to IT Sys Admin: All error messages
4. Status and Error fields in DTO record in BC_CTL. Set on Error or Warning
5. Console screen shown with message: Error messages if DEBUG = 0 or greater, Warnings if DEBUG +1, Information if DEBUG = +2

Please note that you need to set DEBIG to a negative value to stop the console screen appearing.

Error and waning messages may be generated within the BCclass or within a BC Interface. The DTO code can also call ME.msg to report Errors, and provide warning and informational messages.

Commentary

Implementing reliable data exchange interfaces between systems takes a lot of work. It is easy enough to write a few lines of SQL code to move data from one system to another. But to make that transfer work reliably 24x7 year in and year out is a very difficult and time consuming process. That is why MilramX exists to minimize the amount of code that needs to be written, debugged and maintained for each data exchange application.

Please see the companion “MilramX Programmers Manual” for details on how to program DTOs and how to develop automated data exchange and other applications using MilramX.

