Introduction to MilramX

Overview

MilramX is a set of software that was originally implemented as a way of rapidly implementing data exchange interfaces between the BellHawk real-time operations tracking software and a wide variety of ERP and accounting systems. The concept was that MilramX would provide over 90% of all such interfaces consisted of code that could be provided in the form of pre-built libraries or automatically generated code.

MilramX was implemented as an automated data mover that can automatically move data between systems and databases, performing data translation in the process. It is now used as the basis for the BellHawk RTX intelligent alerting software, for collecting data from IOT (Internet of Things) devices, such as RFID portals, as well as for exchanging data with EDI (Electronic Data Interchange) and E-Commerce systems.

MilramX consists of a toolset and a framework. The MilramX framework is designed to ensure reliable automatic 24x7 operation of these data exchange applications. The toolset is an add-in to the Microsoft Visual Studio development environment that enables .Net developers to implement their own data exchange applications using the MilramX framework.

The MilramX Product Line

MilramX is part of a technology stack:

- BellHawk BHSDK (Software Developer Kit) - .Net software library to enable external systems to communicate with BellHawk database using ODBC connection – currently available with BellHawk at no addition charge
- BellHawk WSI (Web Services Interface) – provides capability for external systems to remotely communicate with BellHawk over the Internet using SOAP/XML messages - currently available with BellHawk at no addition charge
- MilramX Framework – this framework is licensed by the processor on which it is executed as well as the number of adaptors (connections to external machines). It contains all the capabilities to periodically fetch data updates from a variety of systems and to run DTOs that carry out some action on this data. This includes special version of the BellHawk BHSDK and WSI interfaces.
- MilramX Toolset – this toolset is licensed by the developer and provides the code needed to develop MilramX Framework based applications
- BellHawk SCI (Supply Chain Integrator) –an add-on set of features for the MilramX Framework that enables the MilramX to be used for automated data exchange between two or more systems.

- BellHawk RTX (Real-Time Alerting) – an add-on set of features for the MilramX that enables it to be used for real-time analysis of "Big-Data" and the generation of operational alerts by Email or text-message to people's mobile phones or other devices.

Competitive Comparison

Unlike many other such automated data mover products, which are designed for E-Commerce use in the Cloud, MilramX (BC) is specifically designed for industrial use in the plant. It is based on the use of Microsoft software, which is commonly used in most industrial environments, and is run on Windows computers within the plant. BC can, however, exchange data with both Cloud-based applications and with Linux, Apple, Android and IBM based applications.

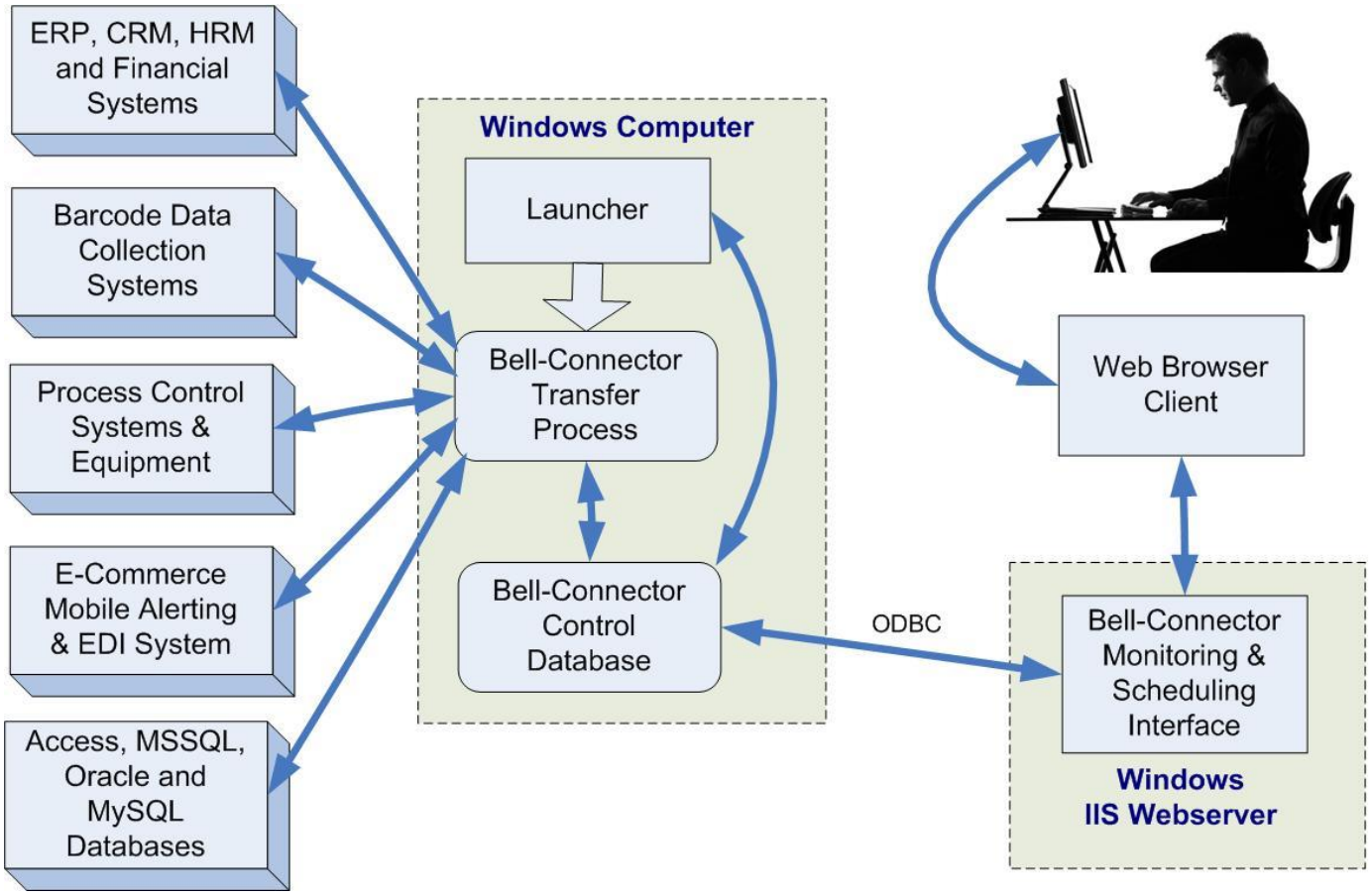
The MilramX toolset is designed for use by VB.Net (Visual Basic) programmers such as IT staff, industrial engineers, and programmer analysts who are not expert software developers. The toolset and framework hides most of the complexity of reading and writing databases and webs-services interfaces from the programmer, enabling them to focus on the business problem at hand.

Much of the work in developing reliable automated data movers between systems is related to handling of errors. It is easy for most programmers to write a SQL script to retrieve data from one database and write it to another. What is hard, and takes over 90% of the code, is detecting and correcting bad data before passing it off to the target system. This may be in the form of bad characters, data formats, or simply data that is supposed to reference some other data item (such as a PO line to a PO) that does not exist in the target system.

BC has extensive built-in mechanisms for detecting and correcting such errors. SCI also has a mechanism for suspending transfers that are not acceptable to the target system. These data objects can then be viewed and edited through the BC web-interface and resubmitted for automated transfer.

A major feature of BC is the periodic scheduling of data transfers and its ability to get just the latest updates from source databases. It also handles multiple levels of indirection in the source database and automatically translates this data into High Level Data Objects (HLDOs), with all the indirections resolved. Similarly it can automatically take a high level business object and resolves multiple levels of indirection as it stores the data in the target system.

Systems Architecture



The MilramX Framework consists of a number of elements:

1. One or more MilramX transfer processes (.exe executable programs) whose job it is to perform the actual transfers between system(s). Within each transfer process there may be one or more Data Transfer Objects (basically named code objects) whose function is to transfer a specific type of HLDO. When the transfer function is executed, the DTO to run is specified in the Transfer functions argument string.

2. A MilramX Control Database – this is a SQL Server database that contains data about the latest time and date each DTO was run and data about when each DTO is scheduled to run next. It also contains tables into which to log failed transfers and, with the SCI, gives the ability to edit and retry failed transfers.

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, with SCI, edit and retry any transfers that failed.

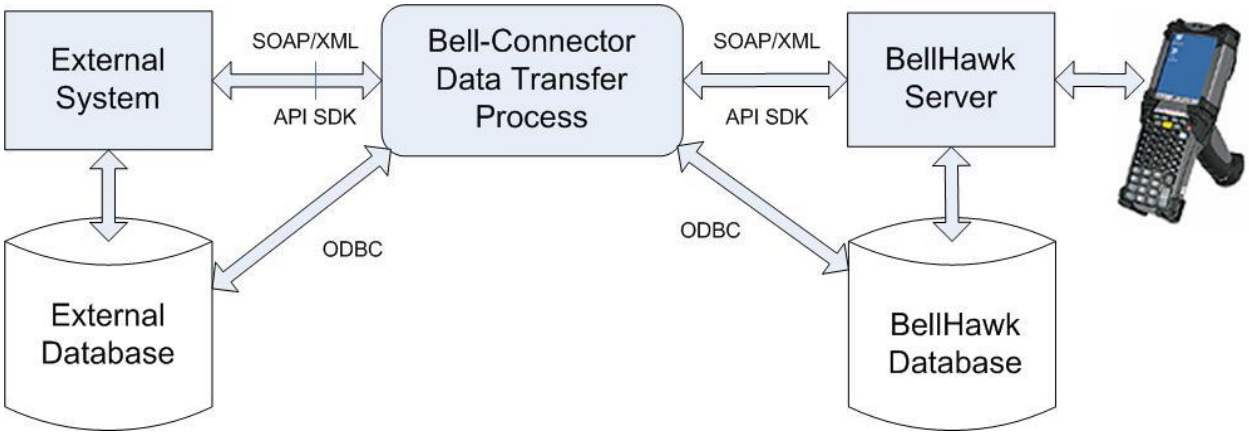
4. A MilramX Launcher program. This Launcher.exe program runs continuously as a background user process. It monitors the setup and status data in the Control database to

decide when to run each transfer process and with which DTO. It also tracks for transfer processes that have exceeded their maximum run-time and kills them if needed.

The launcher and transfer function programs are normally run on a computer in the local plant along with the Control database, which typically uses SQL Server Express. 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 computer used to host the Control database.

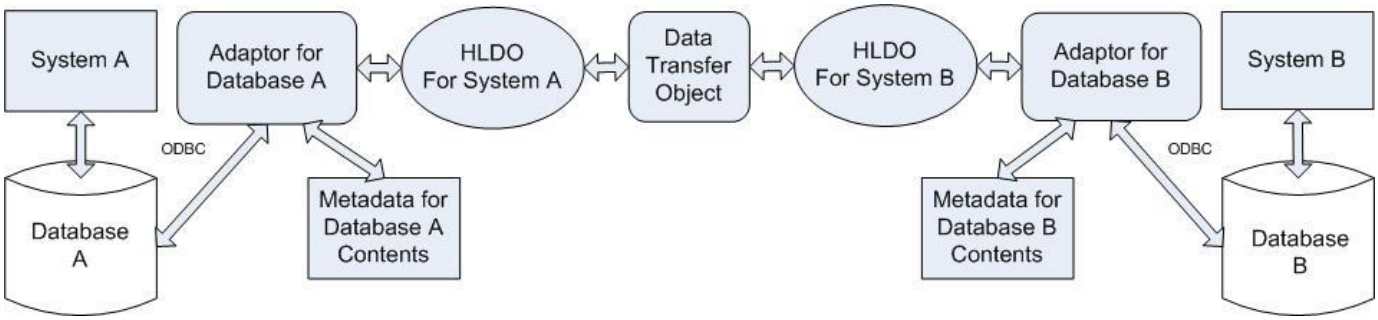
MilramX keeps the last-transferred times separate for each transfer function by tagging these times with the name of the transfer function in the Control database. This enables multiple transfer functions to be active at the same time.

Transfer Process Concepts



The concept of the Transfer process is that it should be able to communicate with multiple external systems directly through ODBC connections or using SOAP/XML web interfaces (or through .Net interfaces to process control systems) . Further that these interfaces would be encapsulated in reusable Adaptors that are able to translate between the complexities of the underlying databases or web-services interfaces and present these to a data conversion layer in the form of High Level Data Objects (HLDOs).

In MilramX, each HLDO has a keyword, such as “Customer” and a set of name-value pairs of parameters such as “Name”:”ABC Co.” The definition of the HLDOs are encapsulated in XML metadata files along with their relationship to underlying tables, views and stored procedures and their fields.



MilramX recognizes that business objects from different systems are not identical and sometimes multiple source business objects are required to produce one target business object. The translation between business objects is written in .Net code and encapsulated in Data Transfer Objects (DTOs), one for each class of HLDO being translated (such as customers).

Adaptors

MilramX uses the concept of Adaptors and BC (MilramX) Interfaces as a way of connecting to different databases and systems. A BC Interface is typically a generic interface, such as code for communicating with Microsoft SQL Server or Oracle databases. Sometimes they are more specific such as an interface to Sage 300 through its .Net SDK, or an interface to QuickBooks Enterprise through QODBC.

Adaptors are BC interfaces combined with information that relates them to a specific database, such as the connection string for the database. This enables the same BC Interface to be used for two different systems (such as BellHawk and Axapta) that both use the MSSQL BC interface but through different Adaptors.

When a transfer function starts executing it reads information about the Adaptors and the BC Interfaces they will use from its initialization (.ini) file. It then uses this information to connect to the databases, web-services, SDK or other interfaces to systems.

BC Interfaces convert generic requests such as Store, Fetch, and Lookup for an HLDO into specific actions to exchange data with a system specified by the Adaptor. While knowledgeable programmers can write their own BC Interfaces, these are mostly written by BellHawk System’s technical staff.

These BC Interfaces encapsulate the details of interfacing with different databases and so reduce the amount of programming work in developing interfaces. They also allow the same interface code to be used with different databases and different ways of interfacing with systems.

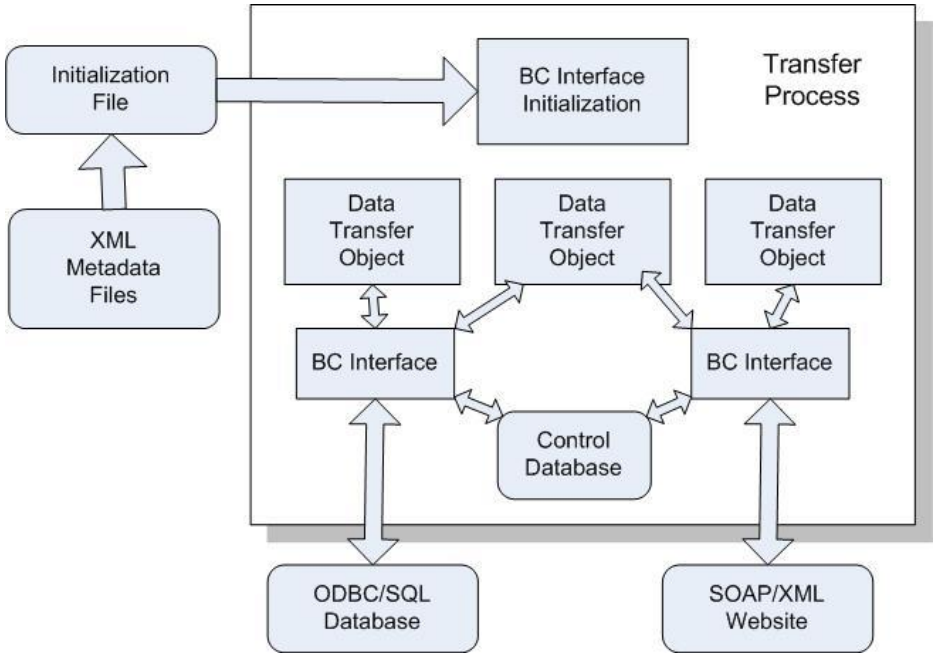
The BC Interfaces currently available as standard with MilramX are:

1. Local direct ODBC access to a Microsoft SQL Server Database. This covers access to a wide range of systems including BellHawk, all of the Microsoft Dynamics ERP systems, and many other ERP systems.
2. Remote access to a BellHawk system using its SOAP/XML web-services interface
3. Local access to QuickBooks Enterprise through the QODBC driver
4. Local ODBC access to an Oracle Database through the Oracle ODBC driver
5. Local Access to Sage 300 through its .Net SDK interface

The following BC Interfaces can be provided by special order:

1. Local ODBC access to a MySQL Database through the MySQL ODBC driver.
2. Local ODBC read-only access to a Sage 100/MAS 90 database
3. Local ODBC access to Microsoft Access through the MDAC ODBC driver

Inside the Transfer Process



Inside the Transfer process there are BC Interfaces that know how to retrieve data from specific systems, such as BellHawk or an Oracle ERP system. The BC Interfaces also know how to store data away into those systems. These BC Interfaces may communicate directly with external systems, such as through their SOAP/XML web services interfaces or through a database connector, that is specific to the database type, such as MSSQL or Oracle.

These BC Interfaces are responsible for checking on the format of the data and making sure that bad data, such a key lookup field containing control characters, do not get passed from one system to another.

These BC Interfaces are used by Data Translation Objects (DTOs), through Adaptors, to retrieve data from one system, translate the data, and then store it away in another system.

When the Launcher launches a Transfer process (there can be multiple), it does so with the index of the DTO entry in the BC_CTL table in the MilramX Control Database. This entry gives the name of the DTO to run, whether to fetch just the latest updates or all of a specific class of business objects, such as all the employee records, or just the latest changes. This database also provides the name of a daily log file into which to log any errors or warnings.

The first task of the Transfer process is to read the .ini initialization file for the Transfer process. This contains the names of all the adaptors, such as "BellHawk" or "Oracle" to be used and the connection information for that database or the web services interface to be used. The Transfer initialization section then uses this to create adaptors and to open the connections to the databases or systems.

In initializing each Adaptor/BC Interface, the Transfer function reads the xml metadata file specified each adaptor. This metafile specifies the keywords for all HLDOs and the parameter names and data types for the HLDOs that can be retrieved or stored through the Adaptor. In the case of databases, the metadata also specifies how the parameters relate to the underlying database tables and fields, views and stored procedures. In the case of SOAP/XML interfaces it specifies the corresponding proxy routine names and the parameters with which they are called. This data is stored away in each Adaptor for use by the DTOs in creating, manipulating and storing away business objects.

High Level Data Objects

At the core of the MilramX is the notion of High Level Data Objects (HLDOs)

An HLDO contains data relating to an instance of an operational entity, such as

- An employee, machine, work center or operation
- Item master part records and related data
- Inventory at a location or stored in a container
- Work orders and Jobs
- Route of operations to be performed
- Bills of materials to be consumed
- Locations where inventory is stored and the facilities in which they are located
- Purchase orders, Purchase Order Lines, and Sales Orders, and Sales Order Lines
- Transactions such as the receiving and shipping of materials or their consumption on jobs or their production by jobs.

One of the problems with most industrial databases is that the data about business objects are stored as trees of indirectly referenced data records in multiple tables. This makes it very difficult to directly translate between data in one system to that in another.

So, in MilramX, we represent HLDOs by:

1. A keyword, such as “ITEM” (for an item master part object) or “EMP” (for an employee).
2. A set of parameter name:value pair strings. For example “PartNumber”:”ABC1234”.

Each HLDO parameter has a data type, such as TEXT, DATE or DECIMAL, which is used for data validation but, other than that, the data representation is in the form of strings which enable the universal translation of data in one system to another

This then enables programmers to write code to translate from a HLDO in one system to that in another, without being concerned with how the data is stored in each system.

This makes it easy to:

1. Write the data translation code
2. Review the code to make sure that it is correct
3. Make subsequent changes

All the details of retrieving and storing these Business Objects are hidden by the adaptors and connectors.

XML Metadata files

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	KEYHEADER	Keyword	Description				ReadWrite	TableName	SrcSelectionCriteria	SrcOrderBy	DtLastUpdateFldName	IsDeletedField	DeletedValue
2		Supplier					ReadWrite	tblVendors	IsDeleted = 0		dtLastMod	IsDeleted	1
3	KEYPARAMS	ParName	Description	FieldType	DefaultValue	IsRequired	ParOrder	Criteria	FieldName	IsIndirect	IndirTableName	IndirInternalRef	IndirExternalRef
4		SupplierNumber		Text		1	1	P	VendorNumber	0			
5		SupplierName		Text		0	2	D	VendorName	0			
6		SupplierCode	Supplier Barcode	TextID		0	3	D	VendorCode	0			
7													

XML metadata files are the key element in translating from raw database structures, which typically have many levels of indirection, to HLDOs (High Level Data Objects) which are easy to translate to other HLDOs, without needing to be concerned about multiple levels of indirection.

An XML metafile starts out as a set of Excel spreadsheets that define the HLDO keywords and parameter names and how they relate to an underlying database or web services interface. These spreadsheets are then converted into an XML metadata file for an Adaptor/Interface using the DEXEL feature of the BC web interface.

Please see the separate MilramX HLDO User’s Manual for details of what the fields in these spreadsheets mean and how to create an XML metadata file from them.

These XML metadata files are placed in the root code directory for the Transfer process and referenced in the Adaptor section of the initialization file.

The interface development concept is that these Excel spreadsheets can be developed by programmers or database administrators who know the database internals of a specific system. As a result you can have different people, with different knowledge bases, providing the metadata for the source and target database.

Then you can have VB.Net programmers, who do not have to know the details of either database, writing the code to translate between the business objects from the source system to those of the target system, without needing to understand the details of where this data is stored in either the source or target databases.

This deconstructs the development process such that it does not need one programmer who knows the intimate details of both systems in order to develop the interface.

Through this metadata mechanism, the adaptors can access tables, views and stored procedures in the system whose business objects are specified in the XML Metadata for that system.

Data Translation Objects

Data Translation Objects or DTOs are code objects that are used to translate from one type of HLDO to another, such as to translate from item master records in one system to another. These are called by the transfer function based on the index into the BC_CTL table, in the BC Control database, with which the Launcher calls the transfer function. These entries in the BC_CTL table specify the name of the DTOs to be called, as well as other settings for running the DTOs which are passed to the DTOS by the transfer function.

DTOs are normally written in VB.Net specifically for each type of data transfer, such as for transferring different types of data object between BellHawk and an ERP system. Typically all the DTOs for one interface are all integrated into one transfer function.

DTOs typically will retrieve all of the latest updates to one type of HLDO from one system and then process them in sequence, storing the results in a second system. Thus, for example a DTO may retrieve all the latest updates to employee records in one system and then process these in sequence before storing each translated data object into a second system.

The intermediate business object format used by the DTOs is an array of named strings, where the names are the parameters of the business object. Each HLDO given a keyword, such as “ITEM” for an item master record. It exchanges data with a named Adaptor.

There is typically one HLDO array of parameter value strings for the source and one for the destination. The primary action of the DTO is to use BCclass functions to read the array of strings from the source data object, translate these in VB.Net code to values in the targetHLDO string array and then to call BCclass to store the resultant data into the target database using the target adaptor.

Some of the functions that a DTO performs are:

1. Lookup a specific instance of a business object, such as the item master record for a specific part number, and get all the parameter values corresponding to that business object instance. This is done by providing the adaptor name and keyword name.
2. Get the set of latest updates to a business object class by keyword in a system referenced by adaptor name.
3. Get the next business object from the retrieved data set and make it accessible as a parameter name:value array.
4. Create an empty business object and move data into its parameters
5. Store a business object using an adaptor for a destination system
6. Delete (or mark as deleted) a business object given the adaptor and keyword.

All these functions, except adaptor initialization, are performed using the BCclass of code objects, as shown in the following snippet of code for a DTO to move suppliers from one system to another:

```
BHVend = New BCclass("Supplier", "BellHawk")

xxVend = New BCclass("XXVEND", "DatabaseXX")

xxVend.SelectRecords("Latest")

While xxVend.NextRecord()

          BHVend.Clear() ' Clear Destination Record

          BHVend("SupplierNumber") = xxVend("Name")

          BHVend("SupplierName") = xxVend("CompanyName")

       BHVend.Store("M")

End While
```

In this snippet, we create a business object instance called BHVend based on the parameters for the keyword “Supplier” in the adaptor “BellHawk”. Then we create a new business object instance based on the keyword “XXVEND” in the adaptor “DatabaseXX”.

Then we select all the latest updates of vendor records from Database XX and index our way through these using the NextRecord class function.

We first clear out any existing data from the BHVend business object and then fill a couple of its parameter values. Finally, we store away the newly updated values for that data object, based on modifying just the fields that have changed (“M” for modify, rather than “C” for change every field in the record).

If a database BC Interface is in use, the Store function looks up the supplier number in the destination table and either does a SQL insert or a SQL update, as appropriate. All the SQL code

is automatically generated by the BC Interface based on the information in the XML metadata file for the HLDO and the initialization file for the Adaptor.

Commentary

For more information, please see the following documents:

1. MilramX Architecture
2. BellHawk Supply Chain Integrator User Manual
3. MilramX Users Manual
4. MilramX High Level Data Object Users Manual
5. MilramX Programmers Manual

