MilramX Implementation Guide

Introduction

This document is designed to give technical guidance to IT staff and developers who are tasked with implementing data transfers between systems and databases using the MilramX toolset.

It is a companion document to:

1. "Installing and Configuring MilramX" – which gives details of how to install and configure a MilramX system on a Windows Workstation or Server computer.
2. "MilramX Users Manual" – which gives details of how to use the MilramX Website interface to setup, monitor, and control data transfers between systems.

Overview

The MilramX (BC) is a generalized automated data mover that moves data between systems and databases performing data translation in the process. It was developed with the goal of providing, or automatically generating, over 90% of the needed code for such automated data movement interfaces, working out-of-the-box. It was further designed to make it easy for .Net developers to use BC to perform the necessary data translations.

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 normally run on computers within the plant. BC can, however, exchange data with both Cloud-based applications and with Linux, Apple, Android and IBM based applications.

While most of the data translation between systems is automated by MilramX, there is not always a nice 1:1 correlation between source and target system data objects. In this case developers can link their own VB.Net (or C#) code into BC to do this translation, while having BC do most of the work of retrieving data from the source system and putting it away in the target system.

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.

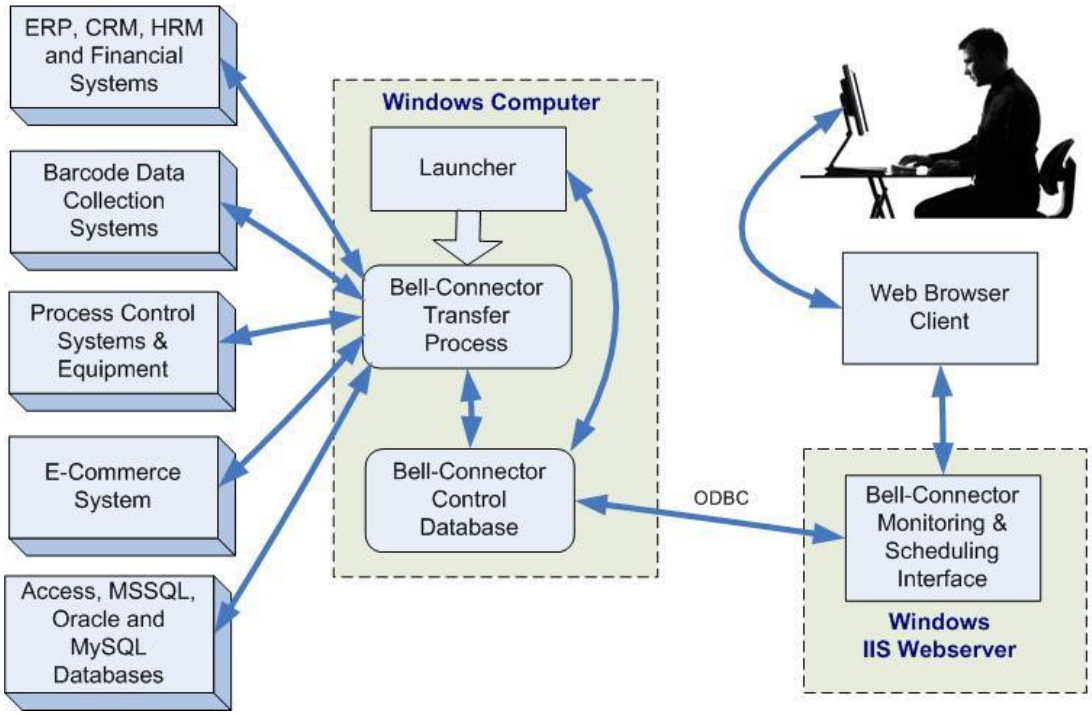
BC has extensive built-in mechanisms for detecting and correcting such errors. It also has a mechanism for suspending transfers that are not acceptable to the target system. These data objects can then be viewed and edited through the BC web-interface and resubmitted for automated transfer.

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 business objects, with all the indirections resolved. Similarly it automatically takes a high level business object and resolves multiple levels of indirection as it stores the data in the target system,

While MilramX is primarily used for data transfers between a source and a target system, it can read and write multiple sources of data in one transfer. It can also be used to interrogate data in multiple systems and to generate alerts, when needed, sent as Text or Email messages to mobile phones and other such devices.

The primary motivation for the development of the MilramX was to enable the BellHawk data collection software to interface with a wide variety of systems. It has a pre-built ODBC adaptor for BellHawk to make it easy to implement these interfaces. We are, however, maintaining and extending MilramX to be a generalized data mover between industrial systems, even when BellHawk is not involved.

Overall Architecture



MilramX 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 the external system(s) and BellHawk. Within the transfer process there may be one or more Data Transfer Objects (basically named subroutines) whose function is to transfer a specific type of data object. When the transfer function is run, the DTO name is specified as one of its arguments.

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 into BellHawk and from which they can be edited and retried. In the DEX2 interface, it contains Import and Export tables with which external applications can exchange data through an ODBC connection.

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.

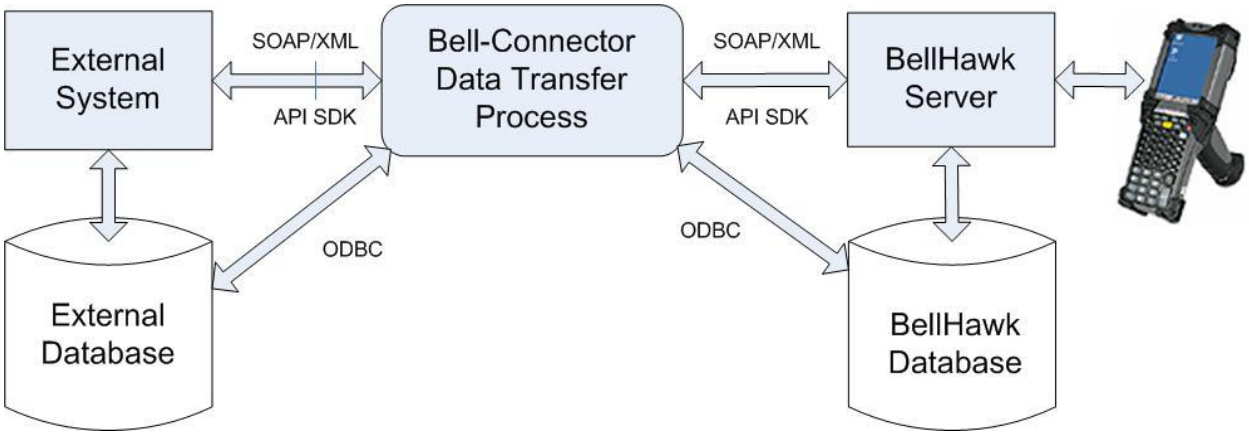
4. A MilramX Launcher program. This Launcher.exe program runs continuously as a background user process. It monitors the setup and status data in the Control database to decide when to run each transfer process and with which DTO. It also tracks for transfer processes that have exceeded their maximum run-time and kills them if needed.

The launcher and transfer function programs are normally run on a Windows 7 Pro computer in the local plant along with the Control database, which typically uses SQL Server Express. They can also be run in a separate user process on a Windows 2008 Server.

The MilramX Website runs under the control of the IIS webserver. It can be installed on the same Windows 7 Pro Workstation computer as the transfer function or it can be run on a separate Windows 2008 Server computer, provided that the Windows Server is on the same LAN or VLAN network as the computer used to host the Control database.

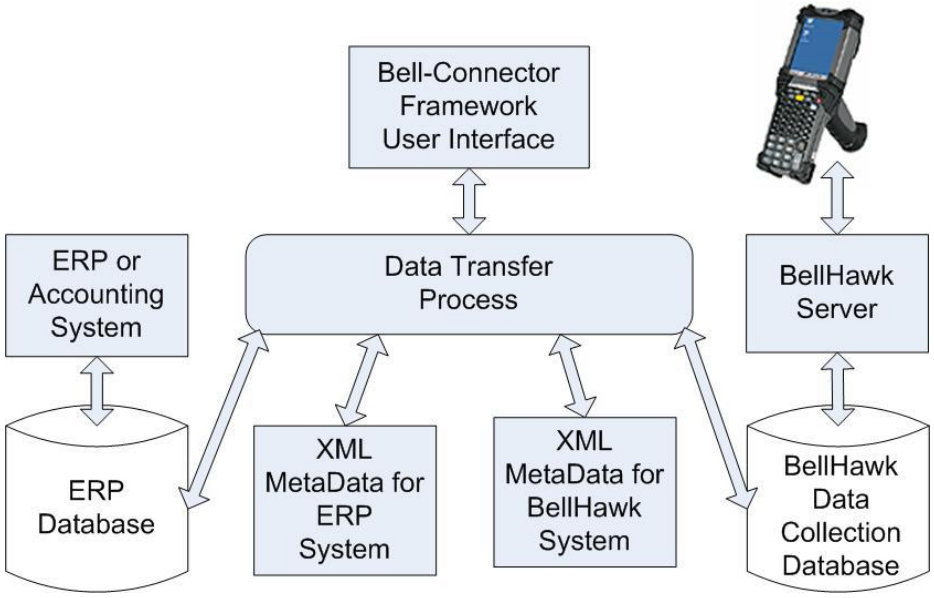
MilramX keeps the last-transferred times separate for each transfer function by tagging these times with the name of the transfer function in the Control database. This enables multiple transfer functions to be active at the same time.

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 simple business objects.

In MilramX, Business Objects has a keyword, such as “Customer” and a set of name-value pairs of parameters such as “Name”:”ABC Co.” The definition of the business objects 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 business objects being translated (such as customers).

Adaptors

MilramX uses the concept of Adaptors as a way of connecting to different databases and systems. These Adaptors 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 Adaptors mechanisms currently available with 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

Other Adaptor mechanisms can be added as needed, including:

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

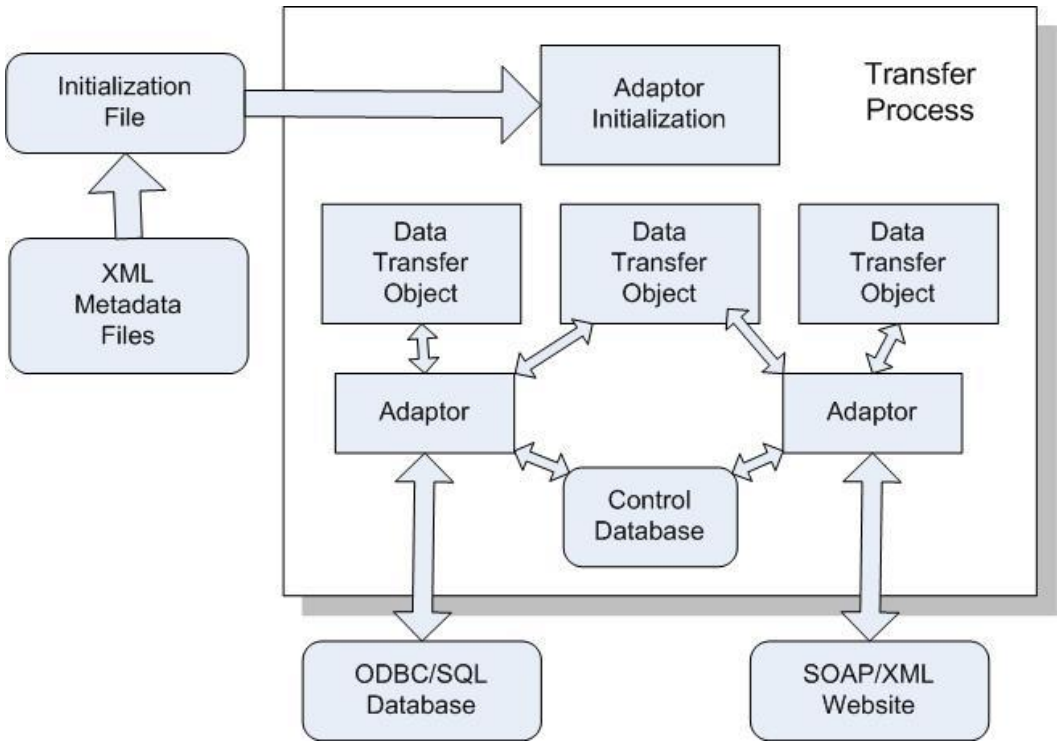
3. Local ODBC access to Microsoft Access through the MDAC ODBC driver

The parameters to each adaptor mechanism, such as database server or web-services connection data, are given in the initialization files for the BC Website and transfer functions, or included in the ODBC DSN (Data Source Name) instance setup.

Generally we do not create adaptor mechanisms for web-services interfaces to external systems. Instead we use their WSDL to create a set of proxy server subroutines that are called within the appropriate DTOs. We made an exception, in the case of BellHawk and implemented an adaptor so that is can be accessed at a remote data center over the Internet in the same way as if it were accessed locally.

Some ODBC drivers, such as the QODBC driver for QuickBooks, are capable of remotely accessing their corresponding systems/databases at remote sites, in the Cloud, over the Internet.

Inside the Transfer Process



Inside the Transfer process you will find "Adaptors" that know how to retrieve data from specific systems, such as BellHawk or an Oracle ERP system. The adaptors also know how to store data away into those systems. These adaptors 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 Adaptors 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 Adaptors are used by Data Translation Objects (DTOs), which 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 creating the adaptor, Transfer reads the xml metadata file specified each adaptor. This metafile specifies the keywords for all business objects and the parameter names and data types for the business objects 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 the adaptor for use by the DTOs in creating, manipulating and storing away business objects.

Business Objects

At the core of the MilramX is the notion of business objects.

A business object 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 the route of operations to be performed
- Jobs and the bill of materials to be consumed
- Locations where inventory is stored and the facilities in which they are located
- Purchase orders and sales orders
- Transactions such as the receiving and shipping of materials or their consumption on jobs or their production by jobs.

Some of these business objects are used to guide the capture of information. Others result from the data capture transactions. Some are used for printing barcode labels and reports or exporting to other systems.

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 business objects 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 parameter has a data type, such as TEXT, DATE or DECIMAL, which is used for data validation but, other than that, the representation is very simple.

This then enables programmers to write code to translate from a business object 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 high level business objects which are easy to translate to other high level business objects, without needing to be concerned about multiple levels of indirection.

An XML metafile starts out as a set of Excel spreadsheets that define the business object 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 using the DEXEL feature of the BC web interface.

Please see the separate WebDexel user's manual for details of what the fields in these spreadsheets mean and how to create an XML metadata file from them.

This 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 the code modules (subroutines) that are used to translate from one type of data objects 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 subroutines to be called, as well as other settings for running the subroutines, which are passed to the subroutines 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 business object 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 business object is given a keyword, such as “ITEM” for an item master record. It is read from a named Adaptor.

There is typically one business object array of 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 target business object 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 connector 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 connector based on the information in the XML metadata file for that adaptor.

Automated Parameter Data Mapping

In many cases of data translation many of the parameters in the source data object have a 1:1 correspondence with those in the target data object. While it is possible to write DTOs with

many 1:1 assignments of source parameter fields to target parameter fields, this can take much unnecessary software development effort.

Instead, users can load an Excel mapping file into the BC's own metadata using the DEXEL functionality of the BC Website and then save it in the BC metadata file so it becomes permanently available.

This mapping metadata, keyword MAP, specifies for a given DTO and source and target adaptor/keyword combinations, how to map the parameters from one data object to another.

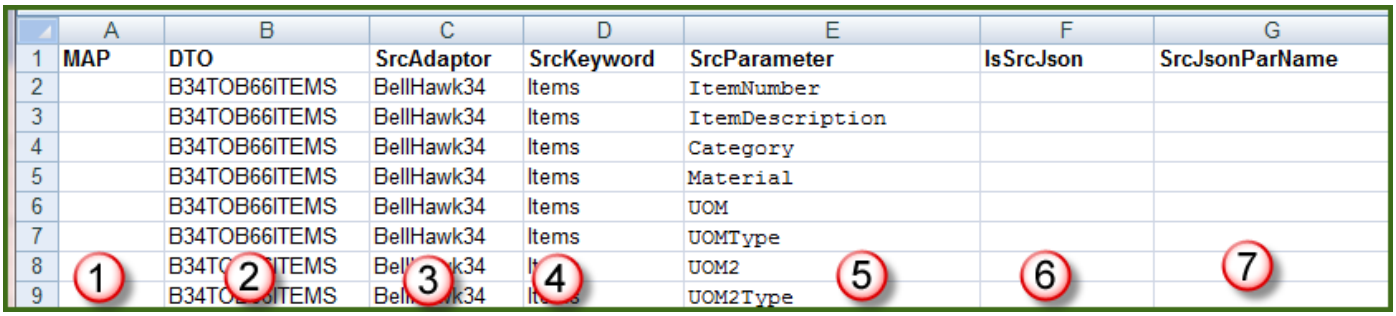
Data from these mapping definitions then be used to automatically transfer data by calling the following subroutine within the DTO:

```
MapObjectParams(DTO_Name, Src_Adaptor_Name, Src_Keyword, Src_Bus_Object,
Tgt_Adaptor_Name, Tgt_Keyword, Tgt_Bus_Object)
```

This function will then move all the specified parameters from the source business object to the target business object without the need to write all the individual assignment statements. This function returns True if the transfers are made without error and False if errors occurred.

This subroutine can then be followed, within the DTO, with code to translate those parameters which do not have a 1:1 mapping, before storing away the resultant target business object.

The format of the Excel spreadsheet to setup automated mapping is shown below:



	A	B	C	D	E	F	G
1	MAP	DTO	SrcAdaptor	SrcKeyword	SrcParameter	IsSrcJson	SrcJsonParName
2		B34TOB66ITEMS	BellHawk34	Items	ItemNumber		
3		B34TOB66ITEMS	BellHawk34	Items	ItemDescription		
4		B34TOB66ITEMS	BellHawk34	Items	Category		
5		B34TOB66ITEMS	BellHawk34	Items	Material		
6		B34TOB66ITEMS	BellHawk34	Items	UOM		
7		B34TOB66ITEMS	BellHawk34	Items	UOMType		
8		B34TO ITEMS	Bell k34	It	UOM2		
9		B34TO ITEMS	Bell k34	It	UOM2Type		

H	I	J	K	L	M	N
TgtAdaptor	TgtKeyword	TgtParameter	IsTgtJson	TgtJsonParName	TgtJsonDataType	Comments
Hampden66	Items	ItemNumber				
Hampden66	Items	ItemDescription				
Hampden66	Items	Category				
Hampden66	Items	Material				
Hampden66	Items	UOM				
Hampden66	Items	UOMType				
Ha n66	Item	UOM2				
Ha n66	Items	UOM2Type				

This metadata specifies how to move data between corresponding fields in like objects, such as Item master records, as shown here.

The meanings of the columns are:

1. MAP (1) identifies this as data to be imported into the Mapping table in the MilramX Control database.
2. DTOName (2) DTO Name – uniquely identifies the data transfer. This may or may not correspond to a DTO subroutine in the MilramX transfer function.
3. SrcAdaptor (3) Source Adaptor Name, such as BellHawk34 – but it could be BellHawk66 if this same mechanism is being used to export reporting data.
4. SrcKeyword (4) Source Keyword – corresponds to a source object keyword, such as Items, for item master records, in the source database XML metadata
5. SrcParameter (5) Source Parameter Name – name of one of the parameters defined for the source keyword in the source database XML data.
6. IsSrcJson (6) (Y/N) – this was added to allow export of UDP data in V6 to an external reporting database. It specifies that the source parameter value is in a JSON encoded string.
7. SrcJsonParName (7) JSON Parameter Name in source UDP field.
8. TgtAdaptor (8) Target Adaptor Name – BellHawk66 for example, but may, in future, be name of the adaptor for a reporting database.
9. TgtKeyword (9) Target Keyword Name - corresponds to the destination object keyword in the destination database XML metadata
10. TgtParameter (10) Target Parameter Name - name of one of the parameters defined for the destination keyword in the destination database XML data.
11. IsTgtJson (11) (Y/N) – Specifies that the data is to be placed in a JSON encoded format in a UDP field in the destination database.
12. TgtJsonParName (12) JSON Parameter Name in target UDP string.
13. TgtJsonDataType (13) TEXTID, TEXT, FLOAT etc. – sets format for destination data in UDP string.
14. Comments (14) – to be used during metadata development for additional comments.

The mapping allows for parameters, encoded in JSON formatted text fields with multiple parameters per text field, to be translated to regular columns in a table, and vice versa.

Here is an example of the use of the UDP Fields

	E	H	I	J	K	L	M
1	SrcParameter	TgtAdaptor	TgtKeyword	TgtParameter	IsTgtJson	TgtJsonParName	TgtJsonDataType
24	IncrementBuildQty	Hampden66	Items	IncrementBuildQty			
25	BuildYield	Hampden66	Items	BuildYield			
26	LotControl	Hampden66	Items	LotControl			
27	Width	Hampden66	Items	UDP	Y	Width	Float
28	Length	Hampden66	Items	UDP	Y	Length	Float
29	Caliper	Hampden66	Items	UDP	Y	Caliper	TEXTID
30	BasisWeight	Hampden66	Items	UDP	Y	BasisWeight	FLOAT
31	IsVocItem	Hampden66	Items	UDP	Y	IsVOC	YNBOOL
32	Color	Hampden66	Items	UDP	Y	Color	TEXTID
33	SalesCaliper	Hampden66	Items	UDP	Y	SalesCaliper	FLOAT
34	SFI	Hampden66	Items	UDP	Y	SFI	YNBOOL
35	FSC	Hampden66	Items	UDP	Y	FSC	YNBOOL
36	PEFC	Hampden66	Items	UDP	Y	PEFC	YNBOOL
37	Normal	Hampden66	Items	UDP	Y	ISNormal	YNBOOL

Automated Character Conversion

One major problem in translating between text strings from a source system to text strings in a target system is that the source system may contain characters which are not acceptable within the target system. Some of these characters, such as embedded control characters, may need to be ignored, while others may need to be translated to an acceptable string of characters for the target system.

These character translations can be specified on an adaptor by adaptor basis by specifying these in an Excel spreadsheet as shown below and adding these to the BC metadata using DEXEL:

	A	B	C	D	E	F	G
1	CHARS	IDNum	Adaptor	InputChar	InputHexCode	OutputChars	AllowInText
2		1	BellHawk34	#	23	lbs	Y
3		2	BellHawk34	°	BA	degrees	N
4		3	BellHawk34	%	25	percent	Y
5		4	BellHawk34	½	BD	1/2	N
6		5	BellHawk34	¼	BC	1/4	N
7		6	BellHawk34	¾	BE	3/4	N
8		7	BellHawk34	&	26	And	Y
9		8	BellHawk34	"	22		Y
10		9	BellHawk34	'	60		Y
11		10	BellHawk34	{	7B		Y
12		11	BellHawk34	}	7D		Y
13		12	BellHawk34	[	5B		Y
14		13	BellHawk34	]	5D		Y
15		14	BellHawk34	(	28		Y
16		15	BellHawk34	)	29		Y
17		16	BellHawk34	|	7C		Y
18	1	2	3	4	5	6	7
19							

These character translations are applied to each text string as the business object is retrieved by the specified adaptor from the appropriate database and before they are made available to the DTO for translation to the target business object.

The meaning of the columns in the spreadsheet are:

1. CHARS (1) is the metadata keyword, which will be used to import this data into the character translation table in the Bell-Converter Control database.

2. The unique IDNum (2) number will be used to identify specific characters on subsequent Excel imports so the entries can be edited or deleted. Entries will retain this IDNum number, even after they have been marked as deleted.

3. The Adaptor column (3) specifies the source of the data and relates to the section of the same name, such as [BellHawk34] in the initialization file used by the transfer function doing the import.

4. The InputChar field (4) is an extended ASCII description of the input character. The actual translation will be performed using the InputHexCode (5) to lookup the character being translated.

5. The output will be translated differently depending on whether the field is designated at a Text type field or a TextID (lookup) field. If the field is defined as a Text field and AllowInText (7) is set to Y for yes then the character will simply be passed through into the text field. Otherwise it will be translated to the character string shown in the OutputChars column (6).

Automated DTO Transfers

As a further savings in software development, users do not have to have to code DTOs where all the work of mapping between data objects and all the character conversion can be setup as described in the previous two sections.

In this case an AutoTransfer DTO can be scheduled to automate these transfers, without needing to do any coding for these pseudo-DTOs.

These AutoTransfers can be setup using an Excel Spreadsheet which is imported into the BC metadata using DEXEL. The format of this spread sheet is:

	A	B	C	D	E	F	G
1	**AUTOTRANSFER**	**SeqNum**	**DTO**	**SrcAdaptor**	**SrcKeyword**	**TgtAdaptor**	**TgtKeyword**
2		1	B34TOB66ITEMS	BellHawk34	Items	Hampden66	Items
3		2	B34TOB66ISTEP	BellHawk34	ISTEP	Hampden66	ISTEP
4							
5	1	2	3	4	5	6	7
6							
7							

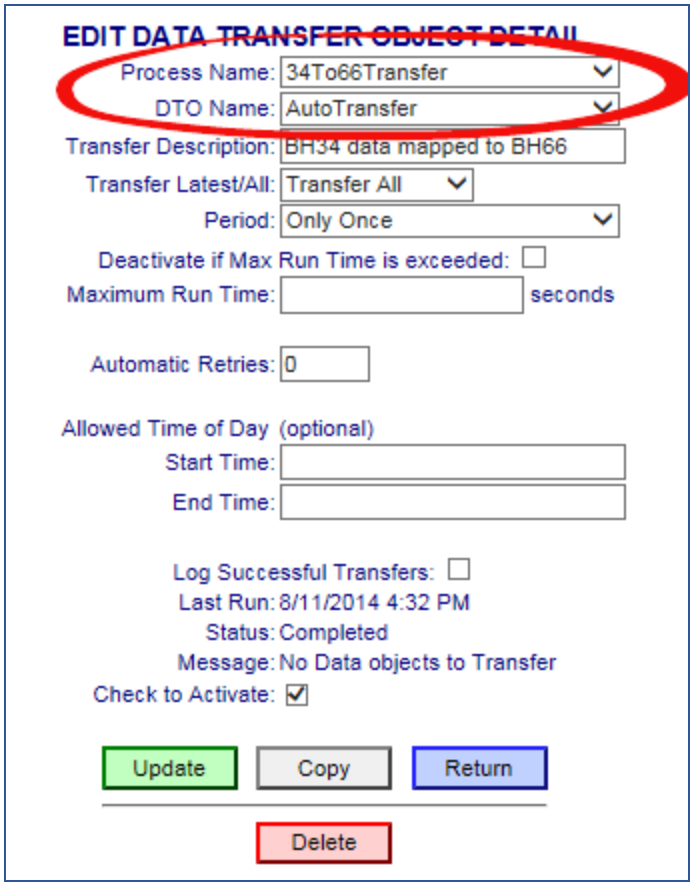
The meanings of the columns are:

1. AUTOTRANSFER (1) – keyword name for metadata controlling the import of this data into the AutoTransfer table in the MilramX Control database using DEXEL.

2. SeqNum (2) – sets the sequence in which the transfers are performed.

3. DTO (3) – sets the DTO name to be used to identify this transfer. There will be no corresponding DTO subroutine in the MilramX transfer function. Nor will there be a separate DTO with this name specified in the available DTOs in the MilramX website. The All/Latest transfers and when the transfers will be performed will all be controlled by the settings for the AutoTransfer DTO.

4. SrcAdaptor (4) and SrcKeyword (5) and the TgtAdaptor (6) and TgtKeyword (7) are passed to the subroutine responsible for performing the mapping.



When using the Auto Transfer function, there is no need to write DTO subroutines or to setup the automatically transferred DTOs through the web-services interface.

All that is needed is to setup the AutoTransfer DTO through the BC Web interface, as shown at right. Then all the pseudo DTOs listed in the Auto-Transfer list will be transferred with the setup parameters (such as transfer latest updates or all records) setup for the Auto-Transfer DTO.

For a situation in which all transfers are performed automatically, only two DTOs need to be setup, as shown below:

	A	B	C	D	E	
1	**DTODEFINITION**	**ProcessName**	**DTOName**	**ParName1**	**ParName2**	**P**
2		34To66Transfer	AutoTransfer			
3		34To66Transfer	RETRY			
4						

The first is the auto transfer DTO and the second is the Retry DTO, to handle any rejected data transfers.

In this case, what is transferred is entirely specified by the Automated Data Mapping, Character Conversion and Auto Transfer data imported via Excel spreadsheets.

Standard Interfaces

BellHawk

MilramX is provided with a fully implemented interface to BellHawk using the MS SQL ODBC interface. This interface has XML Metadata definitions for a wide variety of data objects, views and stored procedures that enable data to be exchanged with a BellHawk system.

By special order, a compatible interface can be provided, which uses the web-services interface to BellHawk, so that BellHawk can be accessed remotely over the Internet.

QuickBooks Enterprise

A standard set of DTOs is available which enable the transfer of data objects between BellHawk and QuickBooks Enterprise through the QODBC interface. There is no charge for these DTOs, above and beyond the licensing cost for MilramX itself, either in binary or source code format, but there is a charge for customizing these DTOs to meet the specific accounting needs of each client.

Commentary

It is planned to follow this Implementation Guide with a Bell-Connecter software development guide, which will give details of the BCclass functions and worked examples how to use these functions to code DTOs.

