High Level Data Object Overview Manual

Introduction

Data exchange with both BellHawk and MilramX is based upon the use of High Level Data Objects (HLDOs). These are sets of parameter-name:value pairs which are identified by a keyword name. They are mapped into JSON strings in Python and other programming languages.

HLDOs are used for:

1. The Excel Setup Screens in BellHawk – to import and export BellHawk setup data.
2. The DEXEL Setup screens in BellHawk and MilramX – Excel import and export for advanced users.
3. The BellHawk BHSDK - .Net Software Development Kit
4. The BellHawk web-services (SOAP/XML) interface
5. The BellHawk MDEX and RDI interfaces
6. Data exchange with MilramX Data Transfer Objects (DTOs)

High Level Data Objects (HLDOs) abstract all the details of the mechanisms needed to lookup, fetch and store data into a universal format that can be exchanged between systems without regard for the communications mechanisms used or how the data is stored in databases or the operating systems used.

An HLDO consists of a keyword, such as “ITEM” and a set of parameter-name:parameter-value pairs of strings, such as “ItemNumber”:”ABC123”. Because parameter values are expressed as strings, they are independent of the data representation used in the underlying databases or interfaces. By convention we use “CamelBack” notation for parameter names, with no spaces (to avoid mistakes).

Each HLDO has an XML metadata representation that describes how the high level data object relates to the schema used by the underlying database or web services interface. This metadata enables:

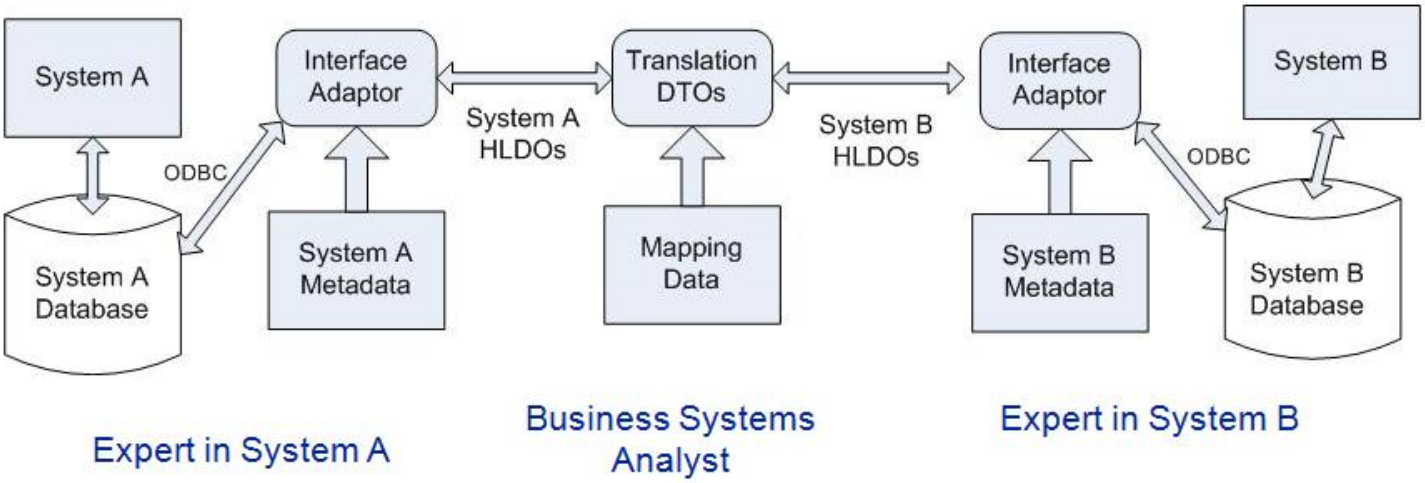
1. BellHawk to automatically generate the SQL code to translate between its underlying database and the HLDO instance data it imports and exports through its Excel interfaces, its BHSDK, or its web-services interface.
2. MilramX to automatically generate SQL code to translate between underlying databases and HLDOs exchanged between systems.
3. Verifying that the data being imported or exporting is in an acceptable format and does not contain any characters (such as control characters) that would be unacceptable to a target system or to code that is processing the HLDO data.
4. Setting of defaults for unknown values when data is stored into a target system.

5. Providing documentation of how HLDOs relate to underlying database, SDK, or web-services interfaces.

The XML metadata for all the HLDOs for a specific system (such as BellHawk) are contained in a single file, such as BellHawk.XML. While XML is supposed to be human readable, on the scale of complex databases such as BellHawk, they can be very difficult to edit or create. As such, we have a variety of mechanisms for creating and editing the HLDO definitions using Excel spreadsheets, which are explained in this manual.

For details of the BellHawk HLDOs, please see the companion manual "HLDO Data Exchange with BellHawk".

Why HLDO Definitions are Important

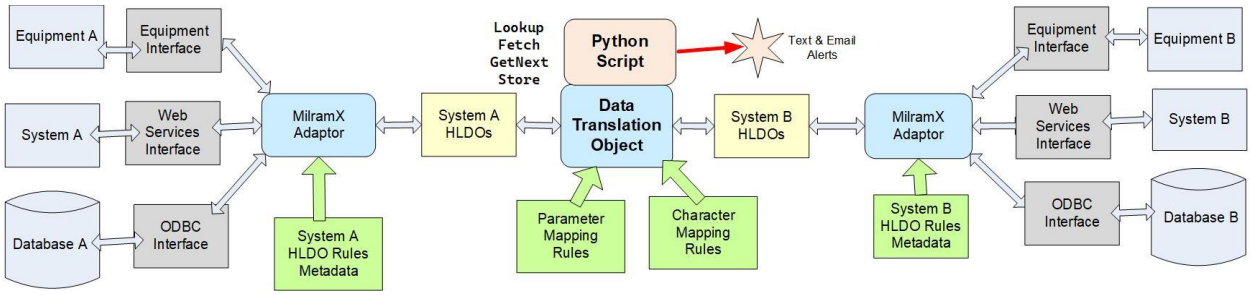


In MilramX, HLDOs play a major role in deconstructing the challenge of developing interfaces between systems. In general, it is relatively easy to find an expert in system A and another expert in system B but, given the thousands of different systems in common use in industrial organizations, it is extremely difficult, if not impossible to find an expert in both systems A and B.

With MilramX, an expert in system A can develop the metadata which translates between the database schema for system A and the HLDOs which can be fetched from system A and stored into system A. Similarly, an expert in system B can develop the metadata to translate between its database schema and a set of HLDOs for system B.

Sometimes there is a 1:1 mapping between the parameters of the HLDOs from one system to those in another, which can be expressed in the form of mapping data that is used by the AutoTransfer Data Transfer Object (DTO) in MilramX, which automatically moves data from system A to system B or vice versa. This mapping is developed in the form of an Excel spreadsheet, typically by a business systems analyst who understand what data needs to be exchanged between A and B.

When there is not a 1:1 mapping between the source and target systems then a DTO written in Python or .Net can be used to translate between the source and target HLDOs.



We have thus deconstructed the impossible problem of finding an expert in both systems A and B to the relatively straightforward problem of getting a team of people with separate knowledge bases to work together using MilramX and HLDOs as the mechanism for collaboration.

In the case of BellHawk, KnarrTek provides all the metadata pre-built, so that the BellHawk HLDOs can be used in BellHawk itself, through its BHSDK and Web Services interfaces, including as part of the MilramX interface to BellHawk.

The XML Metadata

Here is a small segment of the 29,000+ line XML metadata for BellHawk

```
- <tblAdaptorKeys>
      <KeyID>-1</KeyID>
      <AdaptorName>BellHawk</AdaptorName>
      <Keyword>Category</Keyword>
      <Description>Part Categories</Description>
      <TableName>tblItemCategories</TableName>
      <SrcSelectionCriteria>IsDeleted = 0</SrcSelectionCriteria>
      <SrcOrderBy>SeqNo</SrcOrderBy>
      <dtLastUpdateFldName>tmpDtTime</dtLastUpdateFldName>
      <IDPar1/>
      <IDPar2/>
      <IDPar3/>
      <ReadWrite>ReadWrite</ReadWrite>
      <IsDeletedField>IsDeleted</IsDeletedField>
      <DeletedValue>1</DeletedValue>
  </tblAdaptorKeys>
- <tblAdaptorKeys>
      <KeyID>-2</KeyID>
      <AdaptorName>BellHawk</AdaptorName>
      <Keyword>Facility</Keyword>
      <TableName>tblFacility</TableName>
      <SrcSelectionCriteria>IsDeleted = 0</SrcSelectionCriteria>
      <dtLastUpdateFldName>dtLastMod</dtLastUpdateFldName>
      <IDPar1/>
```

While it is possible to create and edit this metadata by hand, this is an extremely difficult, time consuming and mistake-prone process.

To overcome these difficulties, we provide a set of tools to take HLDO definitions in the form of Excel spreadsheets and to convert these into the XML metadata.

The Metadata includes definitions for the HLDO Keywords, as shown above plus definitions for each of the HLDO parameters for each Keyword.

Please note that the Keywords are tied to a specific Adaptor name, in the above example, this is called BellHawk. This is an error detection feature in that the same keyword can appear in the XML metadata for different systems. For example, the Keyword CType (Container Type) may appear in the metadata for both the BellHawk and other adaptors but they have different definitions.

When a link is made, such as by making a call in a MilramX DTO to link the keyword and the adaptor, then the system checks that the specified adaptor for the keyword in the call corresponds to that in the XML metadata file associated with the Adaptor in the initialization file. Note that the Adaptor names can be aliased to match those in the XML metadata in the Adaptor section in the initialization file for the transfer function as described in the MilramX Installation and Configuration Guide.

Please note that while the keywords for multiple adaptors can reside in a single XML metadata file, we have found that it is better to use a metadata file for each Adaptor for each system that MilramX exchanges data with.

Thus, in BellHawk, we have an XML metadata file for the BellHawk ODBC interface (also used by the BHSDK interface) and one metadata file for the BellHawk web-services (SOAP/XML) interface.

Please note that a common Interface, such as MSSQL, can be used to interact with multiple systems, each having their own XML metadata.

Specifying a HLDO in Excel

HLDO definition spreadsheets are used to define the relationship between keywords and parameters and the underlying tables and fields or stored procedures in a database.

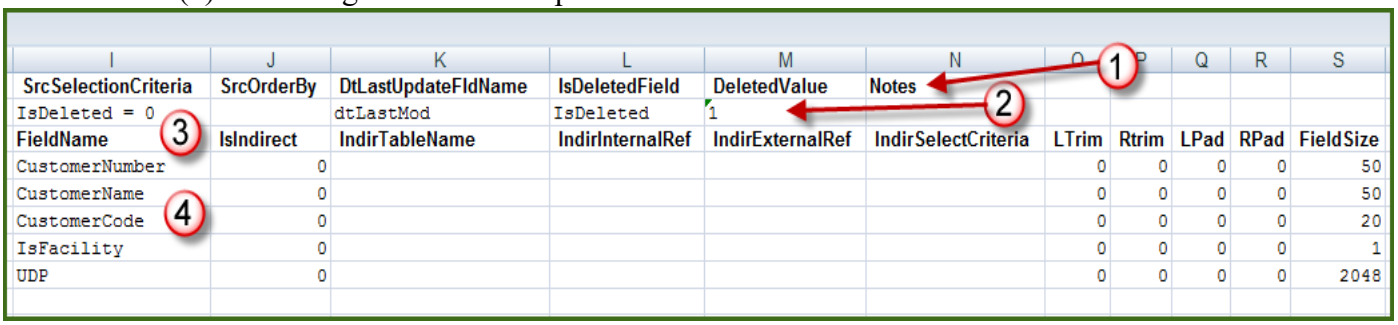
An example of a parameter definition spreadsheet is shown below (in two sections: columns A-H and columns I -S):

Customer.xlsx

	A	B	C	D	E	F	G	H
1	KEYHEADER	Keyword	Description				ReadWrite	TableName
2		Customer					ReadWrite	tblCustomers
3	KEYPARAMS	ParName	Description	FieldType	DefaultValue	IsRequired	ParOrder	Criteria
4		CustomerNumber		TextID		1	1	P
5		CustomerName		Text		1	2	D
6		CustomerCode	Customer Barcode	TextID		0	3	D
7		IsPlant	Is a plant in a multi-plant system	YNBool		0	4	D
8		UDP	User Defined Parameters	JSON		0	5	D
9								
10								

The spreadsheet consists of (from top to bottom):

1. A KEYHEADER row (1), which contains headings for Keyword related items
2. A row containing the Keyword (2) related items
3. A KEYPARAMS row (3), which contains headings for Parameter related items
4. Rows (4) containing data related to parameters.



I	J	K	L	M	N	O	P	Q	R	S
SrcSelectionCriteria	SrcOrderBy	DtLastUpdateFldName	IsDeletedField	DeletedValue	Notes					
IsDeleted = 0		dtLastMod	IsDeleted	1						
FieldName	IsIndirect	IndirTableName	IndirInternalRef	IndirExternalRef	IndirSelectCriteria	LTrim	Rtrim	LPad	RPad	FieldSize
CustomerNumber	0					0	0	0	0	50
CustomerName	0					0	0	0	0	50
CustomerCode	0					0	0	0	0	20
IsFacility	0					0	0	0	0	1
UDP	0					0	0	0	0	2048

The meanings of the KEYHEADER columns are as follows:

Keyword: Specifies a unique keyword to identify the data object.

Description: Description of the data item

ReadWrite: Describes how the parameters are to be treated:

- Read – parameter values can only be read from tables and not written
- Write – parameter definitions that are used to write a table.
- ReadWrite – parameter values can be read and written unless over-ridden by individual parameter
- Run – parameter values relate to stored procedure

TableName: The name of the primary table in the database to which this keyword relates. This can also refer to a database view or a stored procedure name. It may also be used to identify subroutine names in SDKs or other interface objects.

SrcSelectionCriteria: The contents of a SQL WHERE clause (without the word WHERE) that selects specific records from the table. This is typically used to only select active records. Note that the WHERE clause refers to parameter names.

SrcOrderBy: The contents of a SQL ORDER BY clause (without the words ORDER BY) that can be used to order the returned records by default. Note that the ORDER BY clause refers to parameter names.

dtLastUpdateFldName: The name of the field in the primary table that contains the time-date that a record was last updated. This is used for comparison in LATEST and SINCE and RANGE fetches on the data.

IsDeletedField: When a record is "deleted" in the BellHawk database (or many other relational databases), it is not physically deleted but is simply marked as deleted. This is to maintain referential integrity. This column gives the name of the field that is used to mark the record as deleted.

DeletedValue: This is the value that is inserted into the IsDeletedField to indicate that the record is to be considered deleted.

Notes: These are for additional notes about the object definition.

Please note that:

1. There can be more than one keyword and set of parameters defined against the same table. So we might , for example, have one definition for inventoried items and one for non-inventoried items. The keywords have to be unique.
2. There can be separate definitions for reading and writing data. They do need to use different keywords.

The meaning of the `KEYPARAMS` columns are:

ParName: This is the name of a parameter. It must be unique within a keyword. The name consists of alpha-numeric characters but cannot contain any punctuation or non-printing characters. By convention parameter names do not have any spaces. They are case sensitive.

Description: A brief description of each parameter.

FieldType: This information is used to ensure correct formatting of data types written to and read from a database. They are used to control formatting for Excel spread sheet cells. They are also used to verify that the input data does not contain any invalid data for the data type. The available field types are given in the next section.

DefaultValue: Optional. This is to set a default value for a parameter when the HLDO instance is being stored in a database. It is not used when HLDO instance data is fetched from a database.

IsRequired: On Store: if IsRequired = 1 and a value is not present (empty string) but a default value is supplied, the default value is used. If IsRequired = 1 and a value is not present (empty string) and no default value is supplied, then an error is generated. The default value is not used if IsRequired=0 and an empty string parameter value typically results in a Null being stored in the database.

ParOrder: Sets the order in which parameters are exported into an Excel spreadsheet. Also sets order of parameter lookup sequence and order of parameters in call to stored procedure.

Criertia: Sets whether the parameter is a lookup parameter "P" or data "D" for a data table. There may be one or more lookup parameters, such as a Purchase Order Number and Line Number for a purchase order line record. The order in which the parameters are used to uniquely

identify a record is set by the ParOrder entry. While they are order sensitive, “P” and “D” records can be interspersed. “P” is also used to designate the input parameters for a stored procedure call.

If the Criteria field is set to “R” then this is a read-only parameter and will not be written to a table, even if the overall keyword is designated “ReadWrite” or “Write”. This is to avoid attempting to write auto-index key fields in databases and to also avoid attempting to write duplicate field names in the same SQL update or insert statement, which is not allowed by SQL (see below under field names).

For calling a stored procedure the following Criteria can be used:

> “P” an input parameter.
>
> “O” an output parameter
>
> “RW” a read-write parameter – not allowed in SQL Server but allowed by the ODBC specification for access to other relational databases that support read-write parameters.
>
> “RET” for a (singular) return value from the stored procedure

FieldName: Name of field in primary table or view for the object or name of parameter in call to stored procedure (without preceding @sign). Field names do not have to be unique but parameter names do. Thus different parameters can use a common indirect index field to reference different values in a secondary table. But only one parameter of those using a common field name can be marked as having a “P” or a “D” criteria and all the others have to be marked with an “R” to avoid attempts to update or insert records using a repeated field name.

IsIndirect: Sets whether parameter value is directly stored in primary table or accessed through a key to a secondary table, with the key stored in the primary table.

IndirTableName: Contains name of indirect table name – only needed if indirect reference

IndirInternalRef: Name of field in primary table containing ID of record in secondary table. Please note that the name of the ID field in the indirect table does not have to be the same as the ID in the primary table for the HLDO; they just have to contain the same values.

IndirExternalRef: Name of field in secondary table holding actual value of parameter

IndirSelectCriteria : Contents of a where clause for the indirect selection of parameters from secondary table – in terms of field names not parameters.

LTrim: Whether to trim white space characters from front of parameter value. 0= No, 1=Yes.

RTrim: Whether to trim white space characters from end of parameter. 0= No, 1=Yes.

LPad: Whether to add white space characters to front of parameter on writing to database. 0= No, 1=Yes.

RPad: Whether to add white space characters to back of parameter on writing to database. 0= No, 1=Yes.

FieldSize – Maximum number of characters allowed for parameter. Also sets number of bytes after padding on write to database.

The first column of each parameter definition should be left blank.

Note that when metadata for an HLDO is changed in the XML metadata using the Metadata Editor or DEXEL, then the whole of the HLDO metadata is replaced. This is different from importing and exporting HLDO instance data through the BellHawk import or export interface, where placing a D in the first column will cause that specific HLDO data instance to be marked as deleted.

Field Data Types

Every parameter has a specified field data type. These field types are used to check that a data value being set for a parameter is valid; for example that a field type of DATE does indeed contain a string value that can be converted to a date without error. They are also used to check that the value of a parameter being retrieved from the database is valid and does not contain invalid characters for the data type.

The valid field types are:

TEXTID – These are used to specify parameters that are lookup parameters and also text foreign key parameters whose value that must match key values in other tables. These can contain any ASCII character except non printing (control) characters and the percent (%) , comma (,) and double quote (“) characters. Unlike TEXT fields they cannot contain an empty string.

TEXT – Contains any ASCII characters except for non-printing (control) characters. An empty string "" is also a valid TEXT string.

INTEGER – Contains numeric digits. May be prefixed by + or – sign.

FLOAT – Can be prefixed with an optional + or – sign followed by one or more numeric digits. These may be followed by a decimal point and one or more numeric digits. This may be followed by an “e” or “E” for an exponent, followed by an optional + or – sign, followed by one or more numeric digits. An example is 34.79e-12

DECIMAL – May be prefixed by an optional + or – sign, followed by one or more numeric digits, followed optionally by a decimal point and one or more numeric digits. An example is +34.87.

DOLLAR – This is a DECIMAL number that may have an optional $ following the + or – sign. It may also contain commas as separators. The $ sign and commas are stripped out on input and the result is stored in the database in a numeric data format appropriate to the column type in the database. On retrieval the parameter value is returned as a DECIMAL string.

DATE – must be in any valid date format for the system being used. Examples are 10/31/2009 and 2009-10-31.

DATETIME – must be in any valid time data format for the .Net system being used. An example is 10/29/2009 3:35PM.

ID – The ID type is special, in that it is used to identify Auto-Index fields that are generated by SQL Server. These fields can be read from the database but will automatically be excluded from being written when a data object is stored back in the database. These fields will automatically be generated by SQL on a SQL INSERT and reused on an UPDATE.

NUMID – These are used to specify the field types for fields that contain foreign key references to auto ID fields in other databases.

MLTEXT – Multi-Line Text. Same rules as for text except these can contain embedded end of line characters.

YNBOOL – Translate to “Y” or “N” in Excel spreadsheets but are stored in the database as 1 or 0. They can be used to specify the 0/1 no/yes data types found throughout BellHawk.

JSON – a JSON encoded string such as is used for User Defined Parameters in BellHawk.

LIST – a list of comma delimited TEXT values

LISTID – a list of comma separated TEXTID values

Distribution

The HLDO definitions are distributed as an XML metadata file. DEXEL can be used to export the HLDOs in Excel format, from which they can be edited, and then re-imported using DEXEL.

