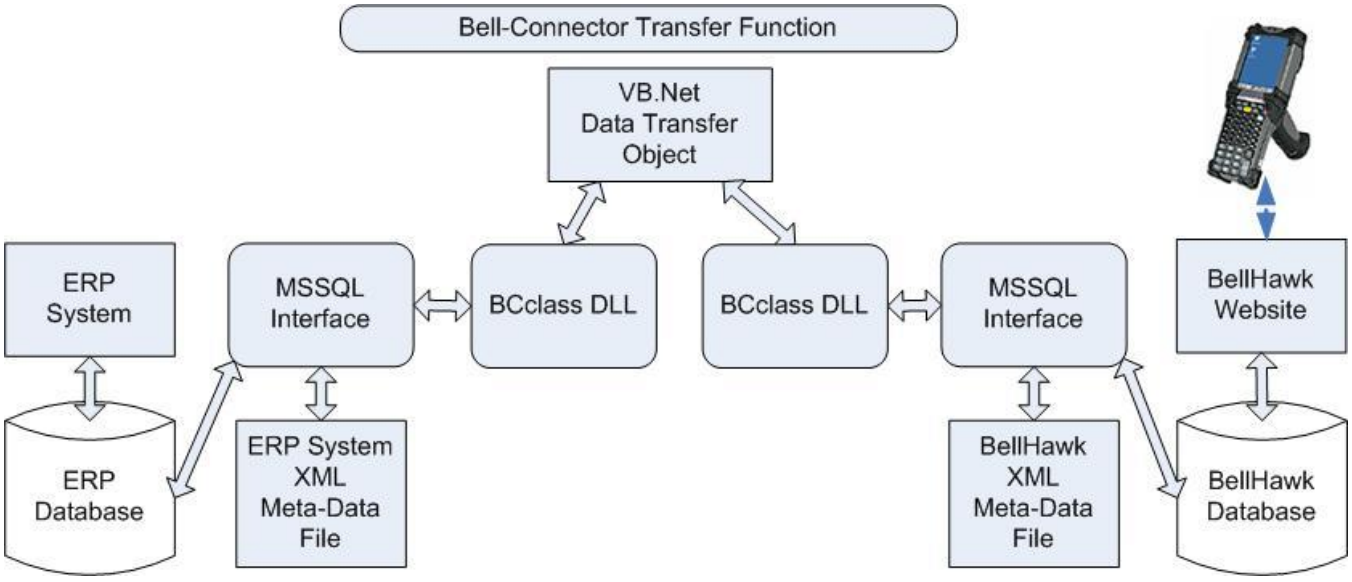
MilramX High Level Data Object User Manual

Introduction



This manual describes how to create and edit the XML metadata files that are used by MilramX, as well as by BellHawk and its BHSDK and Web Services interfaces. This manual describes how to define High Level Data Objects (HLDOs) and to relate these to the software mechanisms used to access databases and systems using software mechanisms such as ODBC, web services TCP/IP messages, and SDK (software development kit) libraries.

The purpose of using HLDOs is to enable programmers to develop application code while being insulated from all the details of the mechanisms needed to lookup, fetch and store data. The HLDO definitions, in the form of XML metadata, enable MilramX to provide generalized interfaces and then to automatically generate SQL code for accessing a wide variety of databases.

The HLDO definitions provide a standard way of fetching and storing data, such that the same application code can be used irrespective of whether the data is being exchanged with a local database or over the Internet through a web-services interface. HLDOs also provide the information for MilramX to check whether data is in the correct format and whether it contains bad characters, such as undesirable control characters embedded in strings.

In MilramX, 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). This makes the keywords and their parameters stand out better in the VB.Net data translation code.

The XML Metadata

Here is a small segment of the 5000+ line XML metadata for a relatively simple MilramX interface to an EDI ASN sending system

```
- <tblAdaptorKeys>
    <AdaptorName>EDICSV</AdaptorName>
    <Keyword>ContainerX</Keyword>
    <Description>Contents of Containers Table</Description>
    <TableName>tblContainers</TableName>
    <SrcSelectionCriteria>IsDeleted = 0</SrcSelectionCriteria>
    <dtLastUpdateFldName>dtLastMod</dtLastUpdateFldName>
    <IDPar1/>
    <IDPar2/>
    <IDPar3/>
    <Notes>Gets contents of containers table</Notes>
    <ReadWrite>ReadWrite</ReadWrite>
    <IsDeletedField>IsDeleted</IsDeletedField>
    <DeletedValue>1</DeletedValue>
  </tblAdaptorKeys>
- <tblAdaptorKeys>
    <AdaptorName>EDICSV</AdaptorName>
    <Keyword>CType</Keyword>
    <Description>Container Types</Description>
    <TableName>tblContainerTypes</TableName>
    <SrcSelectionCriteria>IsDeleted = 0</SrcSelectionCriteria>
    <SrcOrderBy>SeqNo</SrcOrderBy>
    <dtLastUpdateFldName>dtLastMod</dtLastUpdateFldName>
    <IDPar1/>
    <IDPar2/>
    <IDPar3/>
    <ReadWrite>ReadWrite</ReadWrite>
    <IsDeletedField>IsDeleted</IsDeletedField>
    <DeletedValue>1</DeletedValue>
  </tblAdaptorKeys>
```

The XML Metadata for the BellHawk database runs to over 29,000 lines of XML code.

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 EDICSV. 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) appears in the metadata for both the BellHawk and EDICSV adaptors but they have different definitions.

When a link is made, by calling BCclass(keyword, adaptor), in a DTO between 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 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.

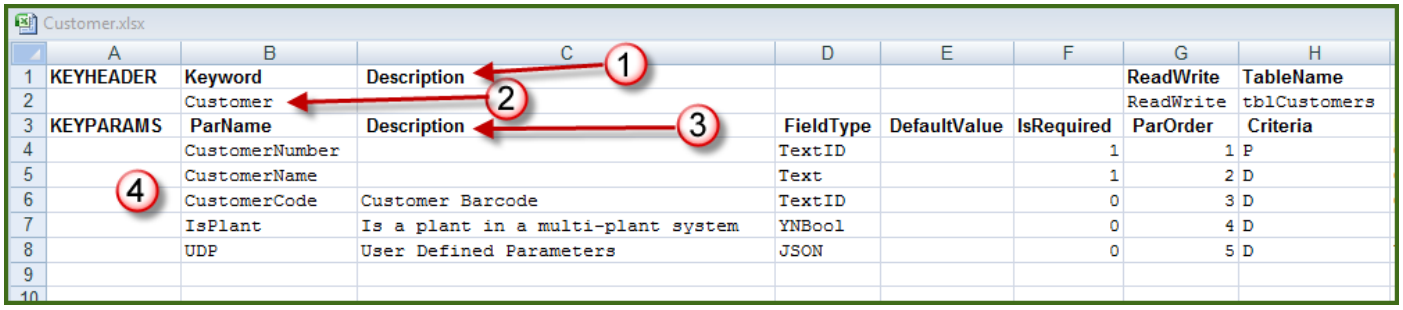
In the above example we used a separate XML metadata file to exchange data with the EDI ASN sending system and elsewhere we use separate metadata files for interfaces to QuickBooks Enterprise and Sage 300.

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

Specifying a HLDO in Excel

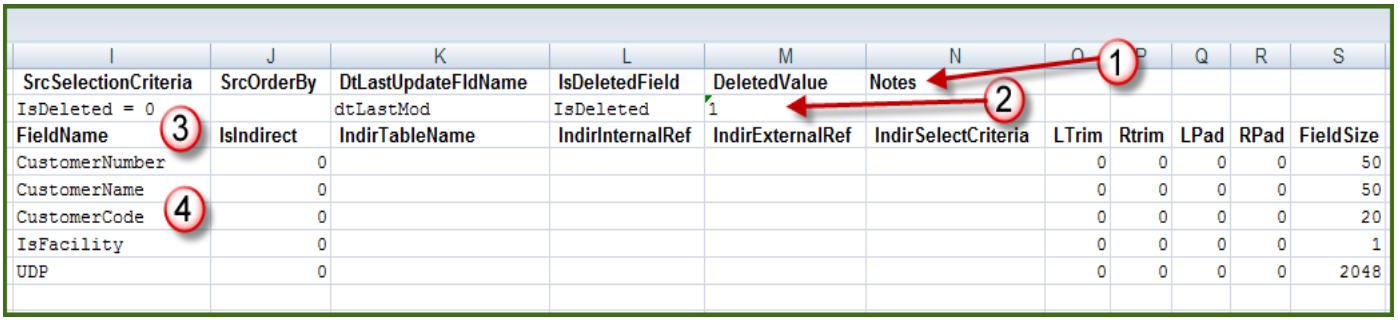
HLDO definition spreadsheets are used to define the relationship between keywords and parameters and the underlying tables and fields or stored procedures in a database.

An example of a parameter definition spreadsheet is shown below (in two sections):



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								



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 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.

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. Normally parameter names do not have any spaces. In release 4, parameter names can have spaces but this feature has not yet been thoroughly tested.

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: If the input data-cell value for a parameter in an Excel spreadsheet being imported is empty then this is the value substituted on import.

IsRequired: Set to 0 if the field is not required to be defined on import and set to 1 if it is required. Note that the import of an object with a required parameter that does not have a default value and that has not been given a value will result in DEXEL generating an error.

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

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 unless the parameter is to be deleted from the metadata. In this case a letter "D" should be placed in column 1, when the existing record for a parameter will be marked as deleted and will not appear in the metadata. Note that this editing is keyed off the parameter name, so if you make a mistake with the name you have to delete the prior parameter entry and add a separate new entry.

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.

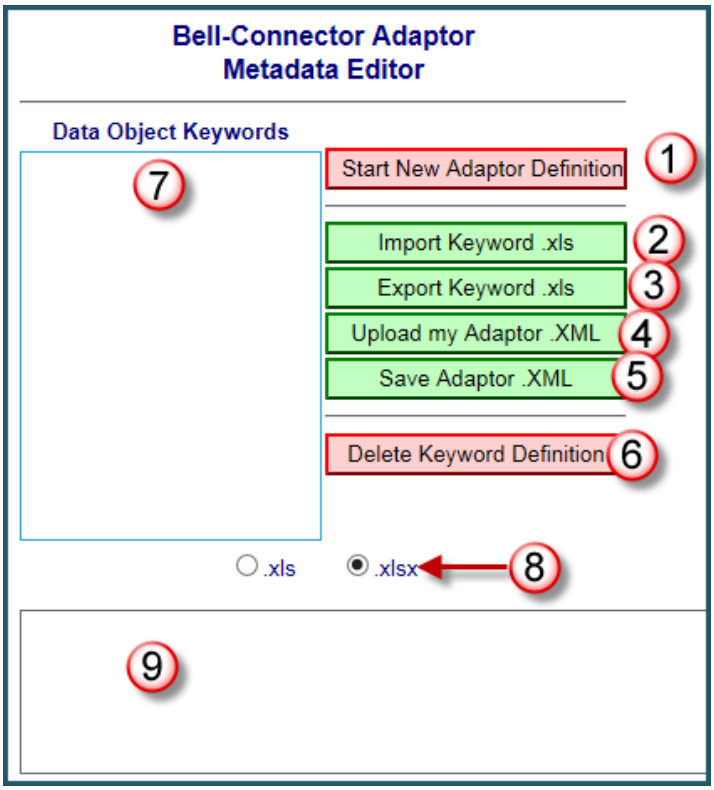
MLTEXT – Multi-Line Text. Same rules as for text except these can contain embedded end of line characters.

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.

These specifications can be converted into XML Metadata files using the Metadata Editor and can then be edited and tested using the DEXEL function of the MilramX Website.

Metadata Editor



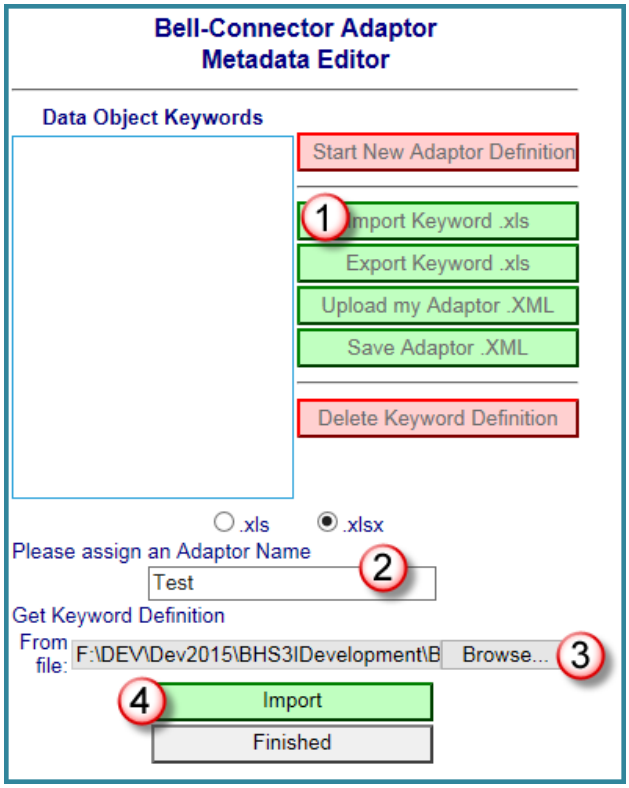
The MilramX Metadata Editor is used to create and edit XML metadata files for use with MilramX. It can also be used to view the Metadata for HLDOs by exporting their definitions in the form of a spreadsheet.

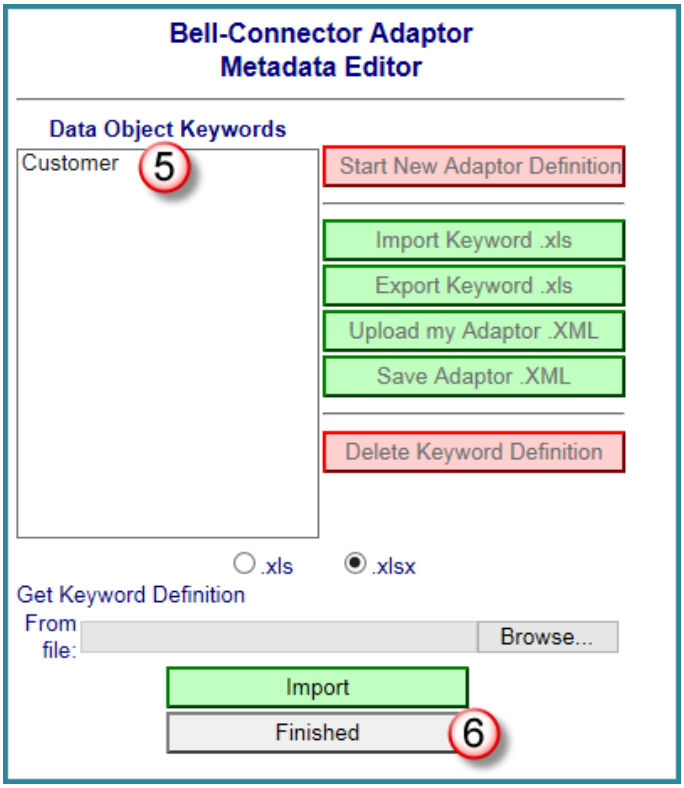
This metadata editor is provided as part of the MilramX developer's toolset. It is installed as a website that, when started shows the screen shown here, which has the following features:

1. A button to create a new adaptor. This starts with a clean XML file. The adaptor name and XML file name are requested when the first HLDO definition is imported.
2. A button (2) to import an Excel definition for a HLDO to be added to the adaptor or to be edited by importing a new HLDO definition file. As each HLDO is added, then its name appears in a list (7) of data object keywords. Any errors in the import are reported in panel (9).
3. A button (3) to export the definition of the HLDO selected in the panel (7) in the format selected by the radio buttons (8).
4. A button (4) to upload a previously saved XML metadata file into memory for editing.
5. A button (5) to download and save the XML metadata from memory.
6. A button (8) to Delete the Keyword Definition, selected in Panel (7), from the XML metadata in memory.

Please note that the Metadata Editor keeps the metadata in memory, so it is important to save the metadata before exiting your web browser session.

Also please note that when a keyword definition is replaced, the whole of the definition is replaced with the new definition. So, if parameters are removed from the Excel import, they will be removed from the metadata created from the import.





When the user selects the Import Keyword button (1) for the first time then the Metadata Editor requests an Adaptor Name (2) and allows the user to browse (3) to the Excel file defining the HLDO and then select the Import button (4).

After this, the keyword for the data object imported is shown in the data panel (5) and the request for the Adaptor Name is no longer visible.

On subsequent imports of HLDOs the Adaptor Name is not requested

Selecting Finished (6) hides the filename selector and Import buttons and reactivates the other buttons, which are grayed out during import.

Recommended Process for Developing, Testing and Deploying Metadata

Introduction

This is a checklist of suggested steps in creating and using a metadata file for an ERP that will exchange data with BellHawk (or some other system).

Recommended Process

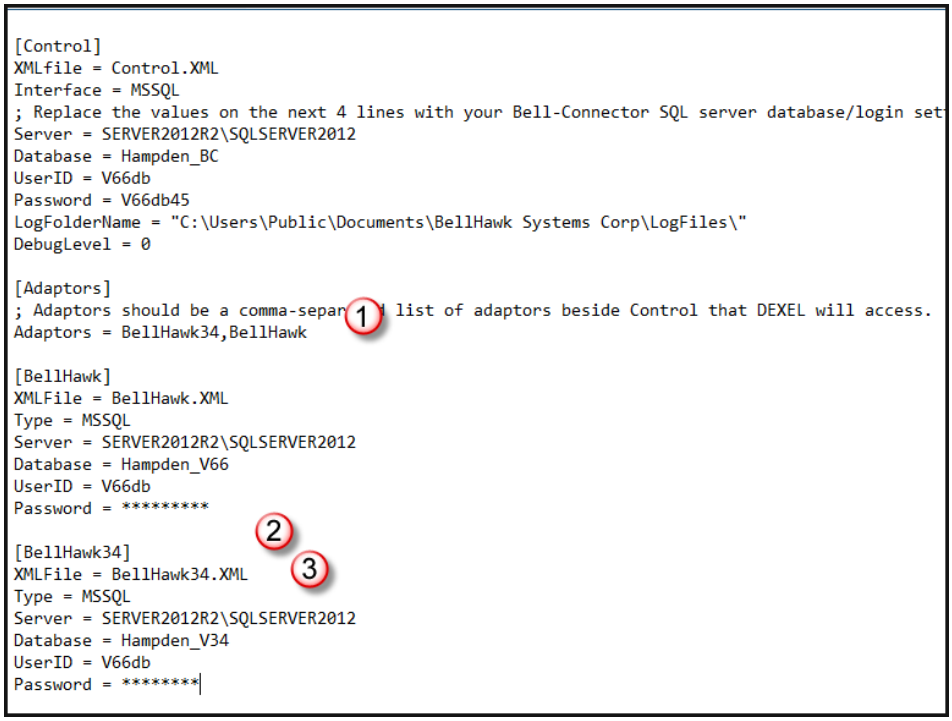
1. Develop Excel spreadsheets for each of the HLDOs that you wish to import or export from your system. This is often best done by starting with the corresponding BellHawk HLDO spreadsheets and then editing these.

2. Use the Metadata Editor (http://MetadataEditor.BellHawkol.com/) to create the adaptor for your system by importing the Excel spreadsheets for the HLDOs for your system and then saving the resultant XML file.

3. Place the XML metadata file you created with the metadata editor into the main BC website folder. (Typically located at C:\inetpub\wwwroot\BCWebsite\).

4. For development purposes, make sure that the BC website folder is readable and writable by members of the IIS_IUSRS group. This will enable you to save changes made to your metadata made through the DEXEL page of the BC website.

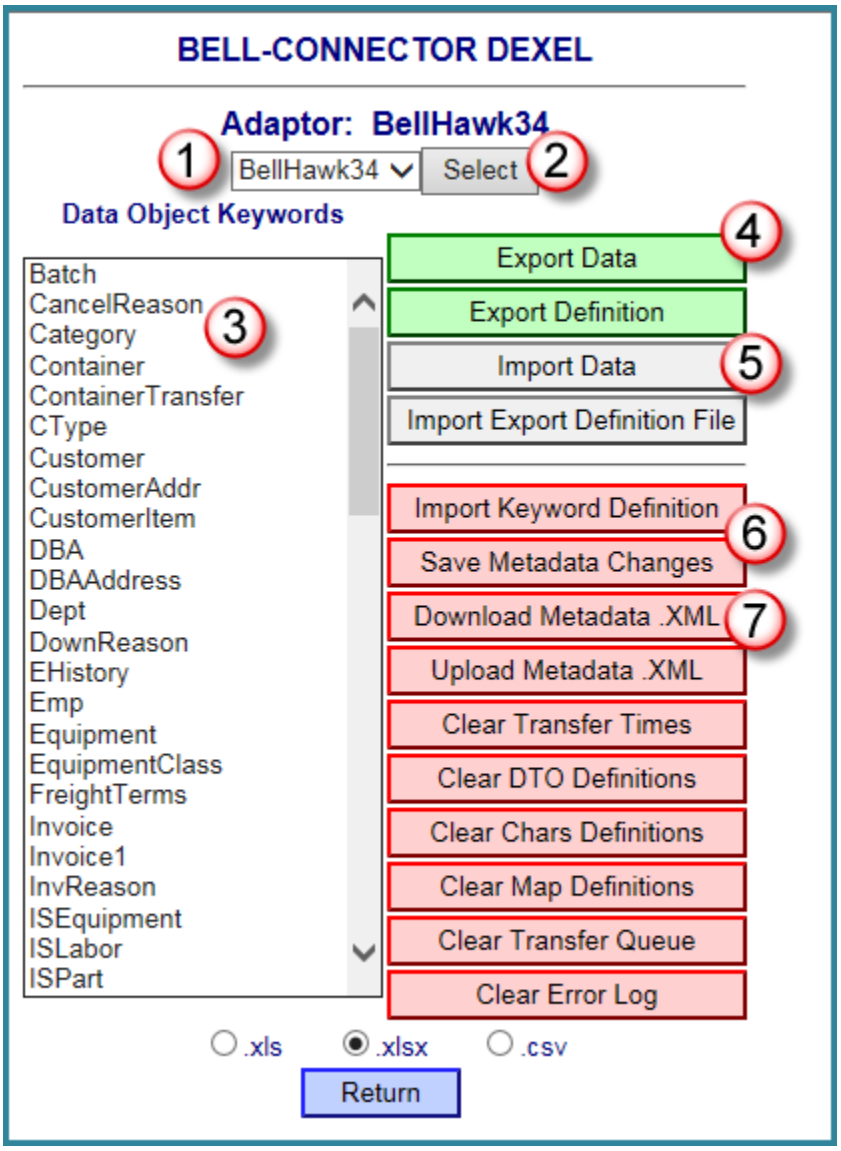
5. Add your adaptor to the initialization BCWebsite.ini file for the BC Website.



6. In the above example we have developed an Adaptor named BellHawk34 to import data from an older V34 version of BellHawk into the latest version of BellHawk. First add your adaptor

to the list of Adaptors and then add a new adaptor section with your Adaptor name (2) and with a reference (3) to the XML metadata file you have created.

7. Restart the browser session you are using to connect to the MilramX website and go to the Admin/DEXEL page. On this page you should be able to:



(1) Select your Adaptor from the drop-down list and then select the [Select] button to load the XML metadata for your adaptor into memory. The HLDOs for your Adaptor should appear in the left hand panel (3).

If they do not appear of you get an error message then probably the values set in the adaptor section for your adaptor in the BCwebsite.ini file are wrong. Also look in the LogFolder (set in the Control section of the ini file for today's error log file) for more detailed error information.

Once you can see your HLDO keywords, select one in the panel (3) and select export data (4). This will export your HLDO data into an Excel spreadsheet, so you can make sure that you are exporting the correct data.

Alternately, if you can test imports of data from an Excel spreadsheet using the Import Data button (5).

You do not have to go back to the Metadata editor to edit your HLDO definitions. You can use BC DEXEL to export and save a definition, edit the HLDO spreadsheet, and then import it again (6) for repeated testing.

Please bear in mind that the metadata changes are only to the metadata in memory. They will not be saved to the metadata file in the main BC website folder until you select [Save Metadata Changes] when it will be saved (providing you have given write permission to BC website folder to the IIS_IUSRS group).

It is usually a good idea, once you have a good working copy of your XML metadata, to also download it (7) and to save it in a safe place.

The only reason to go back to the Metadata editor is if you want to delete an HLDO from the metadata. In this case you would use this editor to upload your XML file, delete the HLDO, possibly add a replacement HLDO, and then save the XML metadata.

8. Once you have tested your metadata file using DEXEL then copy the XML file and the ini file from the BC website folder to the BC Transfer Function director folder.

9. You can now use your metadata in DTOs that you write, or in pseudo DTOs that automatically map parameters from the HLDO for one adaptor to the HLDO for another Adaptor.

Commentary

Developing the Metadata for a system is straight forward but does require following the above process.

Please remember that, if you subsequently change the metadata for your system, using the BC website, to also copy the changed metadata XML file to the BC Transfer function, where it is used to control the exchange of data with your system.

