MxPclass User Manual

Introduction

The MxPclass (MilramX Python Class) Dynamic Link Library (DLL) is used by Python scripts. These scripts are used to code the actions of MilramX and WIPtracker IAPs (Intelligent Agent Processes). This library is used to access the Adaptors and other functions provided by MilramX.

The purpose of the functions provided by this library is to isolate Python script developers from having to know the details of how data is stored in the databases or having to develop SQL Server code.

Also, by using these functions the resulting Python Decision Support code is much simpler, enabling it to be written and understood by business analysts.

Functions

MxPclass Initialization

This does all the MxPclass setup based on the contents of the initialization file

```
MXP = MxPinit(<IAP_path>)
```

Here "<IAP_path>" is the path to the folder holding the main Python Script. MxPinit will use this to get to Control.ini for adaptor definitions.

A Python script will call IAP_path = os.path.abspath(__file__) during its initialization phase and then pass this as the argument to IAPinit( ) as in IAPinit(IAP_path)

After initialization, the MxPinit class can provide some additional information about the condition under which the script was launched.

`MXP.RequestType` returns "All" or "Latest"

`MXP.IsLaunched` returns *true* if it has been launched by the Launcher or *false* if not

`MXP.GetControl` returns the ID of the Control database schedule table row that triggered launch of the script

`MXP.ODBCConnectionString(<AdaptorName>)` returns the connection string for the adaptor <AdaptorName> defined in Control.ini

HLDO Instance Class Initialization

This ties an HLDO (High Level Data Object) keyword to a named Adaptor to create an HLDO instance class. This must be done after MxPclass initialization

```
HLDOinst = MXPclass("Keyword","AdaptorName") # HLDO Adaptor Class Instance
```

Fetch

Fetches a set of HLDO instance records. Returns count of records found.


```
HLDOinst.Fetch(RequestType)
```

Normally Fetch is called with a single Request Type parameter, which can have values of “All” or “Latest”, which gets all records of the specified class that have changed since the last call to Fetch( ). MilramX maintains a date-time at which each HLDO (by Adaptor and Keyword) was last read in a Fetch call.

Fetch( ) can have a number of parameters, besides “All”:

1. “Since” followed by a second parameter, which is the date and time, expressed as a string in standard date-time format. This fetches all records since the specified time-date.
2. “Range”, which is followed by the starting and ending date-times. All records that were modified between these times are retrieved by Fetch(). If the first parameter is set to “” then this acts as “Before” the second parameter. If the second parameter is omitted, then this is the same as “Since” in action.
3. “Specific”, which can be followed by “lookup” parameters values (in the lookup order specified in the XML meta-data file) that uniquely identify the item being fetched.
4. “Select” which is like “Specific” except some lookup parameters name arguments can be set to an empty string `""`. For example, purchase order lines have two lookup parameters, the PO number and the line number. Calling “Select” with just the PO Number as the first parameter will fetch all PO lines for the specified PO number as “Select” assumes that any parameter values not specified contain the empty string by default

Please note that Fetch( ) takes strings as its arguments, so time-date objects need to be converted to strings before calling Fetch. Also Fetch only works on HLDOs defined as Read or ReadWrite.

Get Record

```
HLDOinst.GetRecord(n)
```

Gets the nth record from the most recent Fetch on HLDOinst (0 to count -1) as a JSON string, which can be converted into a Python Dict, as in:

```
HLDOdict = json.loads(DEXinst.GetRecord(n))
```

Next Record

```
HLDOinst.NextRecord()
```

Gets next record from most recent Fetch as a JSON string. Returns an empty string “” if there are no more records available.

Lookup

```
HLDOinst.Lookup(Par1, Par2, . . .)
```

Looks up a specific HLDO instance, based on up to three lookup parameters. Returns an HLDO instance as a JSON string. The number of parameters must correspond to the number of lookup parameters in the HLDO definition.

This is useful for checking whether an HLDO instance, such as an Item Master Part Number, is defined before transferring a PO line that depends on this Item Number.

Store

Stores the current instance of the HLDO in the database table specified by the Adaptor with which the HLDO instance is associated.

```
HLDOinst.Store(json.dumps(HLDOdict), "type")
```

Store takes two parameters:

1. A JSON string containing the HLDO instance values. This is usually created from the HLDO dictionary instance using json.dumps( ).
2. A parameter specifying the type of update “C” or “M”

Store will use the lookup parameters for the HLDO instance to see whether the instance is already in the database table. If it is not already present then Store will insert a new record in the database table. If a matching record is present then it will either replace the whole record, if the second parameter is “C”, or only change records for which the JSON parameter values are not an empty string if the second parameter is an “M”. This latter allows for selective updating of database records.

Store( ) returns True if the values in the HLDO temporary store for the object is successful and False if an indirect data reference could not be resolved for the value provided (such as an unknown PO number in a PO Line object, which is referenced by both PO and PO Line).

Store only works for an HLDO defined as being ReadWrite or Write. It does not work for an HLDO defined as being Read Only or to run a stored procedure.

Run Stored Procedure

```
HLDOinst.Run(json.dumps(HLDOdict))
```

Works just like Store() except that it calls the stored procedure defined in the HLDO definition to output the HLDO JSON string instance.

Running stored procedures to fetch data records is currently not supported.

Errors

```
HLDOinst.ErrDetail()
```

Returns the latest error message for actions on the HLDO instance as a string or an empty string if there is no error. More detailed errors can be found in the daily log file in the folder specified in the initialization file.

Errors should only be printed out if the script is run interactively.

Errors, warnings, and information can also be logged in the log file as in

```
MxMessage.Msg(ErrLevel, Txt)
```


Here the values of ErrLevel can be:

0 = Information

1 = Warning

2 =Error

and Txt is a sting containing the message to be logged.

How these messages are treated will depend on the setting of the MilramX Debug Level in the web interface System Parameters screen, accessible to the administrator.

Verbose output	– Log all output to console and logfile
Errors and Warnings	– Log both errors and warnings to console and logfile
Errors Only	– Log all errors to console and logfile – the default

Sending Email Alerts

A MilramX email alert can be posted using the following call:

```
MXPInit.SendSMTP("AlertName", <message text>, <attachment filename>)
```

When access to an SMTP host has been configured, all subscribers to the alert “AlertName” will be sent an email with the posted message and (optional) attachment file.

Directly Accessing BellHawk and Control Databases

For complex operations sometimes it is easier to directly access local databases. This can be done by using the pyodbc module, with a connection string derived directly from the inifile thus:

```
import pyodbc
dbConnection = pyodbc.connect(MXPInit.ODBCConnectionString(<AdaptorName>))
```

