2019

EFAL

PITNEY BOWES SDK FOR MAPINFO TAB AND MAPINFO ENHANCED TAB FILE ACCESS

PITNEY BOWES SOFTWARE

PITNEY BOWES INC. | 350 Jordan Road | Troy, NY 12180 USA

About the EFAL API

EFAL

The EFAL API consists primarily of one header file named EFAL.h. The API uses only simple C primitive types and function signatures to enable broadest and simplest integration into mixed environments. Most "objects" manipulated by the API such as session, tables, cursors, etc are referenced in API functions as handles. Handles are defined to be unsigned 64 bit integers. Note that handle values are entirely opaque to clients, thus even though some languages such as Java do not support unsigned types, as long as the handle is retrieved and passed back in a way that does not alter its value (e.g. as a signed long integer in Java) the value will be interpreted properly within the SDK. All string values passed across the API are UTF-16 wchar_t strings.

This header file references a pre-processing identifier named "__EFALDLL__" which is used by Pitney Bowes when building the SDK. This identifier should not be defined by any client using the SDK.

The EFAL SDK enables access to the MapInfo SQL data access engine and provides the ability to open, create, query, and modify data in MapInfo TAB and MapInfo Enhanced TAB file formats. This includes MapInfo Seamless tables. Understanding the API consists of the following topics

- Sessions and thread safety – instantiating a session of the MISQL engine and using multiple threads,
- Tables – opening, creating, and describing MapInfo tables,
- MapInfo SQL – executing Select, Insert, Update, and Delete commands,
- Data - accessing data values from records in a table and supplying data values to MISQL commands,
- Geometry – handling of geometry data values, and
- Error handling – accessing error messages and conditions.

These topics will be covered in the subsections below.

Sessions and Thread Safety

The EFAL data access engine must be initialized in what it calls a "session". Tables are opened within a session and all SQL execution and data access occurs within a session. Sessions are initialized through a call to

```
EFALFUNCTION EFALHANDLE InitializeSession();
```

A session handle is returned. The session handle is required by most other calls to the EFAL API. When finished with a session, it must be properly shutdown using the following call

```
EFALFUNCTION void DestroySession(EFALHANDLE hSession);
```

EFAL sessions are thread safe meaning that multiple sessions may be instantiated and used as necessary on server environments. However, an individual session is not multi-thread safe meaning that only a single thread may access a single session at any one time.

Tables

An EFAL session maintains a set of opened MapInfo tables. When first instantiated, a session contains no open tables. The EFAL API includes functions that allow for opening tables (*.TAB), creating new tables (MapInfo TAB, MapInfo Enhanced TAB, and OGC Geopackage), packing a table, listing the opened tables, obtaining the schema and metadata that describes a table, and creation of MapInfo seamless tables.

```
/* ********************************************************
* Table Catalog methods
* ********************************************************
*/
EFALFUNCTION void CloseAll(EFALHANDLE hSession);
EFALFUNCTION EFALHANDLE OpenTable(EFALHANDLE hSession, const wchar_t * path);
EFALFUNCTION void CloseTable(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool BeginReadAccess(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool BeginWriteAccess(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION void EndAccess(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION unsigned long GetTableCount(EFALHANDLE hSession);
EFALFUNCTION EFALHANDLE GetTableHandle(EFALHANDLE hSession, unsigned long idx);
EFALFUNCTION EFALHANDLE GetTableHandle(EFALHANDLE hSession, const wchar_t * alias);
EFALFUNCTION bool SupportsPack(EFALHANDLE hSession, EFALHANDLE hTable, Ellis::ETablePackType ePackType);
EFALFUNCTION bool Pack(EFALHANDLE hSession, EFALHANDLE hTable, Ellis::ETablePackType ePackType);

/* ********************************************************
* Table Metadata methods
* ********************************************************
*/
EFALFUNCTION const wchar_t * GetTableName(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION const wchar_t * GetTableDescription(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION const wchar_t * GetTablePath(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION const wchar_t * GetTableGUID(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION const wchar_t * GetTableType(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION Ellis::MICHARSET GetTableCharset(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool HasRaster(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool HasGrid(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool IsSeamless(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool IsVector(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool SupportsInsert(EFALHANDLE hSession, EFALHANDLE hTable);
```

```
EFALFUNCTION bool SupportsUpdate(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool SupportsDelete(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION bool SupportsBeginAccess(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION long GetReadVersion(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION long GetEditVersion(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION unsigned long GetColumnCount(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION const wchar_t * GetColumnName(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION Ellis::ALLTYPE_TYPE GetColumnType(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION unsigned long GetColumnWidth(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION unsigned long GetColumnDecimals(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION bool IsColumnIndexed(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION bool IsColumnReadOnly(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION const wchar_t * GetColumnCSys(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION Ellis::DRECT GetEntireBounds(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION Ellis::DRECT GetDefaultView(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION unsigned long GetPointObjectCount(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION unsigned long GetLineObjectCount(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION unsigned long GetAreaObjectCount(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION unsigned long GetMiscObjectCount(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION bool HasZ(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION bool IsZRangeKnown(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION Ellis::DRANGE GetZRange(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION bool HasM(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION bool IsMRangeKnown(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);
EFALFUNCTION Ellis::DRANGE GetMRange(EFALHANDLE hSession, EFALHANDLE hTable, unsigned long columnNbr);

/* ********************************************************
* TAB file Metadata methods
* ********************************************************
*/
EFALFUNCTION const wchar_t * GetMetadata(EFALHANDLE hSession, EFALHANDLE hTable, const wchar_t * key);
EFALFUNCTION EFALHANDLE EnumerateMetadata(EFALHANDLE hSession, EFALHANDLE hTable);
EFALFUNCTION void DisposeMetadataEnumerator(EFALHANDLE hSession, EFALHANDLE hEnumerator);
EFALFUNCTION bool GetNextEntry(EFALHANDLE hSession, EFALHANDLE hEnumerator);
EFALFUNCTION const wchar_t * GetCurrentMetadataKey(EFALHANDLE hSession, EFALHANDLE hEnumerator);
EFALFUNCTION const wchar_t * GetCurrentMetadataValue(EFALHANDLE hSession, EFALHANDLE hEnumerator);
EFALFUNCTION void SetMetadata(EFALHANDLE hSession, EFALHANDLE hTable, const wchar_t * key,
                              const wchar_t * value);
EFALFUNCTION void DeleteMetadata(EFALHANDLE hSession, EFALHANDLE hTable, const wchar_t * key);
EFALFUNCTION bool WriteMetadata(EFALHANDLE hSession, EFALHANDLE hTable);

/* ********************************************************
* Create Table methods
* ********************************************************
*/
EFALFUNCTION EFALHANDLE CreateNativeTableMetadata(EFALHANDLE hSession, const wchar_t * tableName,
                const wchar_t * tablePath, Ellis::MICHARSET charset);
EFALFUNCTION EFALHANDLE CreateNativeXTableMetadata(EFALHANDLE hSession, const wchar_t * tableName,
                const wchar_t * tablePath, Ellis::MICHARSET charset);
EFALFUNCTION EFALHANDLE CreateGeopackageTableMetadata(EFALHANDLE hSession, const wchar_t * tableName,
                const wchar_t * tablePath, const wchar_t * databasePath, Ellis::MICHARSET charset);
EFALFUNCTION void AddColumn(EFALHANDLE hSession, EFALHANDLE hTableMetadata, const wchar_t * columnName,
                Ellis::ALLTYPE_TYPE dataType, bool indexed, unsigned long width, unsigned long decimals,
                const wchar_t * szCsys);
EFALFUNCTION EFALHANDLE CreateTable(EFALHANDLE hSession, EFALHANDLE hTableMetadata);
EFALFUNCTION void DestroyTableMetadata(EFALHANDLE hSession, EFALHANDLE hTableMetadata);
```

```
/* ********************************************************
* Create Seamless Table methods
* ********************************************************
* A seamless table is a MapInfo TAB file that represents a spatial partitioning of feature
* records across multiple component TAB file tables. Each component table must have the same
* schema and same coordinate system. This API exposes two functions for creating a seamless
* table. CreateSeamlessTable will create an empty seamless TAB file located in the supplied
* tablePath. AddSeamlessComponentTable will register the specified component TAB file into
* the seamless table. The registration entry will use the supplied bounds (mbr) unless the
* mbr values are all zero in which case the component table will be opened and the MBR of the
* component table data will be used.
*/
EFALFUNCTION EFALHANDLE CreateSeamlessTable(EFALHANDLE hSession, const wchar_t * tablePath,
                const wchar_t * csys, Ellis::MICHARSET charset);
EFALFUNCTION bool AddSeamlessComponentTable(EFALHANDLE hSession, EFALHANDLE hSeamlessTable,
                const wchar_t * componentTablePath, Ellis::DRECT mbr);
```

MapInfo SQL

EFAL uses the MapInfo SQL language for accessing and modifying data within tables. The MISQL language consists of SELECT, INSERT, UPDATE, and DELETE commands. Queries (SELECT statements) return a cursor object as a handle and the API provides functions for looping through the records of the result set as well as accessing the data values for the fields of the current record. INSERT, UPDATE, and DELETE commands return a long integer indicating the number of records that were affected by the command.

Functions that are used for querying and modifying data are

```
/* ********************************************************
* SQL and Expression methods
* ********************************************************
*/
EFALFUNCTION EFALHANDLE Select(EFALHANDLE hSession, const wchar_t * txt);
EFALFUNCTION bool FetchNext(EFALHANDLE hSession, EFALHANDLE hCursor);
EFALFUNCTION void DisposeCursor(EFALHANDLE hSession, EFALHANDLE hCursor);
EFALFUNCTION long Insert(EFALHANDLE hSession, const wchar_t * txt);
EFALFUNCTION long Update(EFALHANDLE hSession, const wchar_t * txt);
EFALFUNCTION long Delete(EFALHANDLE hSession, const wchar_t * txt);

EFALFUNCTION EFALHANDLE Prepare(EFALHANDLE hSession, const wchar_t * txt);
EFALFUNCTION void DisposeStmt(EFALHANDLE hSession, EFALHANDLE hStmt);
EFALFUNCTION EFALHANDLE ExecuteSelect(EFALHANDLE hSession, EFALHANDLE hStmt);
EFALFUNCTION long ExecuteInsert(EFALHANDLE hSession, EFALHANDLE hStmt);
EFALFUNCTION long ExecuteUpdate(EFALHANDLE hSession, EFALHANDLE hStmt);
EFALFUNCTION long ExecuteDelete(EFALHANDLE hSession, EFALHANDLE hStmt);
```

The included MapInfo SQL language reference file (MISQL_Reference.chm) provides more details on the supported language constructs, operators, and functions.

There are two sets of command styles for SELECT, INSERT, UPDATE, and DELETE. The first set accepts an MISQL statement and executes it. This set of commands parse the statement, bind the identifiers to tables, columns, and variables as appropriate, executes the command and finally cleans up the internal parsed statement resources. The second set is intended for optimizing batch operations in which an MISQL command (which may be a SELECT, INSERT, UPDATE, or DELETE statement) is first prepared through a call to Prepare which returns an EFALHANDLE to a statement. This

call performs the command parsing and identifier resolving to tables, columns, and/or variables as appropriate. Subsequent calls to the Execute functions (ExecuteSelect, ExecuteInsert, etc.) accept a previously Prepared statement handle. Typically the prepared statement references variables whose values change between subsequent calls to the Execute commands. Once finished, a statement must be disposed of through a call to DisposeStmt.

Simple statement processing would typically look like this

```
EFAL::CreateVariable(hSession, L"@OBJ");
// Create point WKB and bind it to a variable
const wchar_t * strInsertStatement =
      L"INSERT INTO addresses (StreetAddress, MainAddress, X, Y, OBJ) "
      L" VALUES ('3630 SPINNAKER DR', 'ANCHORAGE, AK  99516-3429',"
      L" -149.829305, 61.10065, @OBJ)";
long nrecs = EFAL::Insert(hSession, strInsertStatement);
```

while the Prepare-Execute model of processing would look more like the following:

```
const wchar_t * strInsertStatement =
      L"INSERT INTO addresses (StreetAddress, MainAddress, X, Y, OBJ) "
      L" VALUES (@StreetAddress, @MainAddress, @X, @Y, @OBJ)";
EFAL::CreateVariable(hSession, L"@StreetAddress");
EFAL::CreateVariable(hSession, L"@MainAddress");
EFAL::CreateVariable(hSession, L"@X");
EFAL::CreateVariable(hSession, L"@Y");
EFAL::CreateVariable(hSession, L"@OBJ");
EFALHANDLE hInsertStatement = EFAL::Prepare(hSession, strInsertStatement);
while (true)
{
      // Set values for variables @StreetAddress, @MainAddress, @X, @Y, and @OBJ
      EFAL::ExecuteInsert(hSession, hInsertStatement);
}
EFAL::DisposeStmt(hSession, hInsertStatement);
```

Data

A cursor returned from a call to the Select function behaves like a typical database "fire-hose" cursor in that it is positioned on a current record and can be repositioned sequentially to the next record until there are no more records in the result set of the original query.

The following functions provide for the ability to determine the cursor's schema (number, name, and data type of the returned columns) and the data values of the respective fields for the current record.

```
/* ********************************************************
* Cursor Record Methods
* ********************************************************
*/
EFALFUNCTION unsigned long GetCursorColumnCount(EFALHANDLE hSession, EFALHANDLE hCursor);
EFALFUNCTION const wchar_t * GetCursorColumnName(EFALHANDLE hSession, EFALHANDLE hCursor,
                  unsigned long columnNbr);
EFALFUNCTION Ellis::ALLTYPE_TYPE GetCursorColumnType(EFALHANDLE hSession, EFALHANDLE hCursor,
                  unsigned long columnNbr);
EFALFUNCTION const wchar_t * GetCursorColumnCSys(EFALHANDLE hSession, EFALHANDLE hCursor,
                  unsigned long columnNbr);
EFALFUNCTION const wchar_t * GetCursorCurrentKey(EFALHANDLE hSession, EFALHANDLE hCursor);
```

```
EFALFUNCTION bool GetCursorIsNull(EFALHANDLE hSession, EFALHANDLE hCursor, unsigned long columnNbr);
EFALFUNCTION const wchar_t *  GetCursorValueString(EFALHANDLE hSession, EFALHANDLE hCursor,
                unsigned long columnNbr);
EFALFUNCTION bool GetCursorValueBoolean(EFALHANDLE hSession, EFALHANDLE hCursor, unsigned long columnNbr);
EFALFUNCTION double GetCursorValueDouble(EFALHANDLE hSession, EFALHANDLE hCursor, unsigned long columnNbr);
EFALFUNCTION long long GetCursorValueInt64(EFALHANDLE hSession, EFALHANDLE hCursor, unsigned long columnNbr);
EFALFUNCTION long int GetCursorValueInt32(EFALHANDLE hSession, EFALHANDLE hCursor, unsigned long columnNbr);
EFALFUNCTION short int GetCursorValueInt16(EFALHANDLE hSession, EFALHANDLE hCursor, unsigned long columnNbr);
EFALFUNCTION const wchar_t *  GetCursorValueStyle(EFALHANDLE hSession, EFALHANDLE hCursor,
                unsigned long columnNbr);
EFALFUNCTION unsigned long PrepareCursorValueBinary(EFALHANDLE hSession, EFALHANDLE hCursor,
                unsigned long columnNbr);
EFALFUNCTION unsigned long PrepareCursorValueGeometry(EFALHANDLE hSession, EFALHANDLE hCursor,
                unsigned long columnNbr);
EFALFUNCTION double GetCursorValueTimespanInMilliseconds(EFALHANDLE hSession, EFALHANDLE hCursor,
                unsigned long columnNbr);
EFALFUNCTION EFALTIME GetCursorValueTime(EFALHANDLE hSession, EFALHANDLE hCursor, unsigned long columnNbr);
EFALFUNCTION EFALDATE GetCursorValueDate(EFALHANDLE hSession, EFALHANDLE hCursor, unsigned long columnNbr);
EFALFUNCTION EFALDATETIME GetCursorValueDateTime(EFALHANDLE hSession, EFALHANDLE hCursor,
               unsigned long columnNbr);
```

Queries may also refer to bound variables. For example, the query

```
SELECT obj FROM states WHERE state_name = @stateName
```

refers to a variable named "@stateName". This variable will be located and bound to the query when the Select statement is executed. An EFAL session also maintains a collection of named variables. Variable names do not have to begin with an @ sign, however, this is good practice to avoid confusion with other identifiers such as column names.

The following functions in the EFAL API provide the ability to create, drop, and list defined variables as well as get and set their values.

```
/* ***********************************************************
* Variable Methods
* ***********************************************************
*/
EFALFUNCTION bool CreateVariable(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION void DropVariable(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION unsigned long GetVariableCount(EFALHANDLE hSession);
EFALFUNCTION const wchar_t * GetVariableName(EFALHANDLE hSession, unsigned long index);
EFALFUNCTION Ellis::ALLTYPE_TYPE GetVariableType(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION Ellis::ALLTYPE_TYPE SetVariableValue(EFALHANDLE hSession, const wchar_t * name,
                  const wchar_t * expression);

EFALFUNCTION bool GetVariableIsNull(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION const wchar_t *  GetVariableValueString(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION bool GetVariableValueBoolean(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION double GetVariableValueDouble(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION long long GetVariableValueInt64(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION long int GetVariableValueInt32(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION short int GetVariableValueInt16(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION const wchar_t *  GetVariableValueStyle(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION unsigned long PrepareVariableValueBinary(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION unsigned long PrepareVariableValueGeometry(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION const wchar_t * GetVariableColumnCSys(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION double GetVariableValueTimespanInMilliseconds(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION EFALTIME GetVariableValueTime(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION EFALDATE GetVariableValueDate(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION EFALDATETIME GetVariableValueDateTime(EFALHANDLE hSession, const wchar_t * name);
```

```
EFALFUNCTION bool SetVariableIsNull(EFALHANDLE hSession, const wchar_t * name);
EFALFUNCTION bool SetVariableValueString(EFALHANDLE hSession, const wchar_t * name, const wchar_t * value);
EFALFUNCTION bool SetVariableValueBoolean(EFALHANDLE hSession, const wchar_t * name, bool value);
EFALFUNCTION bool SetVariableValueDouble(EFALHANDLE hSession, const wchar_t * name, double value);
EFALFUNCTION bool SetVariableValueInt64(EFALHANDLE hSession, const wchar_t * name, long long value);
EFALFUNCTION bool SetVariableValueInt32(EFALHANDLE hSession, const wchar_t * name, long int value);
EFALFUNCTION bool SetVariableValueInt16(EFALHANDLE hSession, const wchar_t * name, short int value);
EFALFUNCTION bool SetVariableValueStyle(EFALHANDLE hSession, const wchar_t * name, const wchar_t * value);
EFALFUNCTION bool SetVariableValueBinary(EFALHANDLE hSession, const wchar_t * name, unsigned long nbytes,
                   const char * value);
EFALFUNCTION bool SetVariableValueGeometry(EFALHANDLE hSession, const wchar_t * name, unsigned long nbytes,
                   const char * value, const wchar_t * szcsys);
EFALFUNCTION bool SetVariableValueTimespanInMilliseconds(EFALHANDLE hSession, const wchar_t * name,
                   double value);
EFALFUNCTION bool SetVariableValueTime(EFALHANDLE hSession, const wchar_t * name, EFALTIME value);
EFALFUNCTION bool SetVariableValueDate(EFALHANDLE hSession, const wchar_t * name, EFALDATE value);
EFALFUNCTION bool SetVariableValueDateTime(EFALHANDLE hSession, const wchar_t * name, EFALDATETIME value);
```

The function setVariableValue assigns a value to a variable using a string expression. The expression can be any valid MISQL expression and can also be a select statement. When the expression is a select statement, the variable is assigned the value of the first column returned from the first record.

Geometry

Geometry values are transmitted across the EFAL API using the OGC GeoPackage binary encoding. This is the only portion of the API that is concerned with bundling multiple values together and thus requires knowledge of the endian order of the representation. The GeoPackage binary specification encompasses this and thus the EFAL API itself does not expose any indication as to the endianness of the interface. The following is a sample snippet for accessing a geometry binary representation from a cursor

```
unsigned long nbytes = EFAL::PrepareCursorValueGeometry(hSession, hCursor, i);
char * bytes = new char[nbytes];
EFAL::GetData(hSession, bytes, nbytes);
```

EFAL currently supports the standard geometry types defined within the standard and MapInfo Custom LegacyText geometry as an extended geometry type (206) (the X bit in the header will be set to 1).

Other MapInfo Custom geometry types such as LegacyEllipse, Arc, RoundRect and Rect will be returned as Polygon or LineString on read.

Coordinate systems are generally returned through function calls as a string. String representations are always in Codespace:Code format. Typically when returned the codespace will be "mapinfo" and the code will be a MapInfo PRJ string like the following:

```
mapinfo:coordsys 1,62
```

EPSG codes may be supplied such as "epsg:4326".

Geometry objects encoded as GeoPackage binary include an SRS_ID field in the header. In general this field is ignored for MapInfo file formats and will be either 0 or -1 when geometries are returned (0 for geodetic coordinate systems and -1 for projected coordinate systems).

If accessing data from an OGC GeoPackage table or inserting data into a GeoPackage table, the srs_id field is meaningful. When reading data from a GeoPackage table, the ID references the internal reference system record, however, the coordinate system can also be obtained from the table or the cursor in Codespace:Code format as previously described. When inserting or updating a geometry value, the caller is responsible for supplying the proper SRS_ID field value in the binary structure. The EFAL API currently does not provide access to the registry to determine the defined coordinate reference systems.

Error Handling and Resources

EFAL does not throw exceptions across the interface boundary. Functions in EFAL return either Boolean success indicator or a primitive data type value for which it should be easy to identify presence of failed calls. For example, the OpenTable function returns a handle. Any function that returns a handle will return zero if there was a failure. EFAL exposes a function to interrogate if an error exists stored within the session, clearing the error state, and also for retrieving the text of the error.

The following functions are supplied for detecting and retrieving errors

```
/* ********************************************************
* Error Handling
* ********************************************************
*/
EFALFUNCTION bool HaveErrors(EFALHANDLE hSession);
EFALFUNCTION void ClearErrors(EFALHANDLE hSession);
EFALFUNCTION int NumErrors(EFALHANDLE hSession);
EFALFUNCTION const wchar_t * GetError(EFALHANDLE hSession, int ierror);
```

Error strings are returned in English only currently. The SDK contains a file named `EFALErrorStrings.properties`. It is possible to use localized string resources with EFAL and Pitney Bowes will eventually supply localized versions of the resources. The EFAL SDK will load this file after first searching a file using the locale name as returned from the Windows SDK function GetUserDefaultLocaleName (e.g. "en-US"). The file it searches for would be EFALErrorStrings_***en-US***.properties. If not found, then the default EFALErrorStrings.properties file will be used.

EFALLIB

EFAL can be bound to your application in two ways – statically through use of LIB files or dynamically through calls to GetProcAddress. The header file EFALLIB.h has been provided to make it easier to use the dynamic binding. Dynamic binding may be preferable in cases where your application needs to be able to start and handle cases where EFAL is not available or if your application needs to handle multiple versions where certain methods may have not yet existed. Using static binding, the EFAL functions would be called like this as an example

```
EFALHANDLE hMaxCursor = EFAL::Select(hSession, szwSelect);
```

Using the dynamic binding, the application would create an instance of the EFALLIB class (which provides stubs for calling GetProcAddress) like this

```
EFALLIB * efallib = EFALLIB::Create(L"EFAL.dll");
```

For dynamic binding, the above example would be written as follows:

```
EFALHANDLE hMaxCursor = efallib->Select(hSession, szwSelect);
```

If your application uses multiple threads, the instance of EFALLIB can be shared across threads but care should be taken to initialize it only once (e.g. with a mutex).

Sample Applications

C++ - EFALShell

EFALShell is a command line application built using the EFAL API. It presents a simple prompt (>) at which commands may be specified. The supported commands are only part of the application and not indicative of the full capabilities of the EFAL API.

The HELP command displays the following

```
> help

Lines beginning with ' or # are comment lines and will be ignored.
A blank line terminates the execution.

Commands:
  Open [Table] <<filename>> [as <<alias>>]
              Opens the specified TAB file. The table will be assigned the default alias or an
              an alias may be supplied.
  Pack <<table>> - Packs the specified table (if pack is supported by that table type)
  Close {<<table>> | ALL} - closes the specified table or all tables
  Set <<var>> = <<expr>> - creates or updates a variable with the value of the expression.
              Expressions may be any valid MISQL expression and may reference other variables
              by name. Expression may also be a SELECT statement in which case the value will be
              from the first column of the first record of the result set.
              Variable names should begin with an '@' sign as good convention.
  Show [<<var>>] - shows all defined variables or a specific variable if specified.
  Tables - lists all currently opened tables.
  Desc <<table>> - describes the specified table. Describe will display the columns, their data
              types, and will also display the metadata from the .TAB file, if any.
  Save [table] <<table>> in <<filename>> [charset << charset >> ][NOINSERT]
  Save [table] <<table>> as nativex table in <<filename>> [UTF16 | UTF8 | charset << charset >> ][NOINSERT]
  Save [table] <<table>> as geopackage table <<dbtable>> in <<file>>
                                    [coordsys <<codespace:code>>]
                                    [UTF16 | UTF8 | charset << charset >> ][NOINSERT]
              Create a new table from an existing table. There are three versions of the Save
              command, the first creates a MapInfo "native" TAB file while the second form
              creates a MapInfo Enhanced ("nativex") file. MapInfo Native files do not support
              UTF-8 or UTF-16 character encodings, MapInfo Enhanced file do.
              The third form is used to create a new OGC GeoPackage table. If the specified database
              file does not exist, it will be created.
              By default the source table's data will also be copied into the new table, however the
              NOINSERT keyword can be used to create an empty table.

  The following commands use the MapInfo SQL language syntax.
  Select ...
  Insert ...
  Update ...
  Delete ...

>
```

The implementation of the various commands illustrates how to utilize the EFAL API. The application initializes a single EFAL session on start-up and destroys that session on shutdown as the wmain function shown below:

```
int wmain(int argc, wchar_t * argv[])
{
      hSession = EFAL::InitializeSession();
      Prompt();
      EFAL::DestroySession(hSession);
      return 0;
}
```

The command processing function makes a call to a function named MyReportErrors after every command. This function shows how to check if there are errors accumulated in the EFAL session and will get those errors and print them to the console.

A SELECT statement command in EFALShell will display to the console the results of the specified query. For example:

```
> select state, state_name, obj from USA where state LIKE 'N%'
MI_KEY  State   State_Name      Obj
28      NE      Nebraska        wkbPolygon[( -104.053009,   40.000000)(  -95.308034,   43.001740)]
29      NV      Nevada  wkbPolygon[( -120.004723,   35.002084)( -114.038800,   42.002298)]
30      NH      New Hampshire   wkbMultiPolygon[(  -72.557682,   42.696896)(  -70.611458,   45.305246)]
31      NJ      New Jersey      wkbMultiPolygon[(  -75.559800,   38.928407)(  -73.894400,   41.357330)]
32      NM      New Mexico      wkbPolygon[( -109.049562,   31.332066)( -103.001479,   37.000274)]
33      NY      New York        wkbMultiPolygon[(  -79.762182,   40.496600)(  -71.856640,   45.012533)]
34      NC      North Carolina  wkbMultiPolygon[(  -84.321949,   33.834212)(  -75.460460,   36.587999)]
35      ND      North Dakota    wkbPolygon[( -104.048424,   45.935061)(  -96.554159,   49.000700)]
8 records
   0.1 seconds
```

Notice that for geometry values, the API returns OGC GeoPackage binary values. The EFALShell sample application embodies only enough parsing logic to determine the geometry type and the MBR for display purposes. More robust clients will need to supply their own full parsing implementation.

C# - NFAL and NFALShell

The solution includes a .NET version of EFALShell named NFALShell. This application is written in C# and relies on a .NET assembly version of the EFAL API. NFAL is the .NET assembly that exposes the EFAL API as an assembly using the C++/CLI language. The command set and functionality of NFALShell is identical to EFALShell. The section below provides an overview of the supplied Visual Studio solution, projects, and dependencies.

Java – JFALShell

The solution includes a Java version of EFALShell named JFALShell. The command set and functionality is identical. The EFAL API was designed to make the integration into other languages such as Java as easy as possible. The solution builds a JNI wrapper over EFAL called JFAL and then uses JFAL to implement the JFALShell application. The section below provides an overview of the supplied Visual Studio solution, projects, and dependencies.

EFAL SDK Distribution and Visual Studio Solution

The EFAL SDK consists of 3 top level folders:

- Data – contains sample data files in MapInfo, MapInfo Enhanced, and OGC GeoPackage formats.
- Export – Contains the EFAL SDK in x64 and Win32 architectures. These are also the folders in which the sample projects will be built into and can be run from.
- Solution – Contains the Visual Studio sample projects. [1] The Solution folder contains a folder named Samples in which the EFAL_Samples.sln Visual Studio solution file is located.

The Samples Visual Studio solution consists of 5 projects: EFALShell, NFAL, NFALShell, JFAL, and Xport.

EFALShell Project

EFALShell contains the C++ code that implements the functionality defined earlier. The project is mostly defined in a single cpp file name efalshell.cpp. The project includes the public EFAL header files from the EFAL\include directory and links to the EFAL DLL using the supplied lib in EFAL\lib (uw32 or ux64).

NFAL and NFALShell Projects

The NFAL project creates a .NET assembly for the EFAL API using C++/CLI. This assembly can be used for any application written for the .NET CLR runtime (C#, VB, or C++/CLI). NFALShell is a project that uses the NFAL assembly that replicates the EFALShell console functionality in C#.

JFAL Project

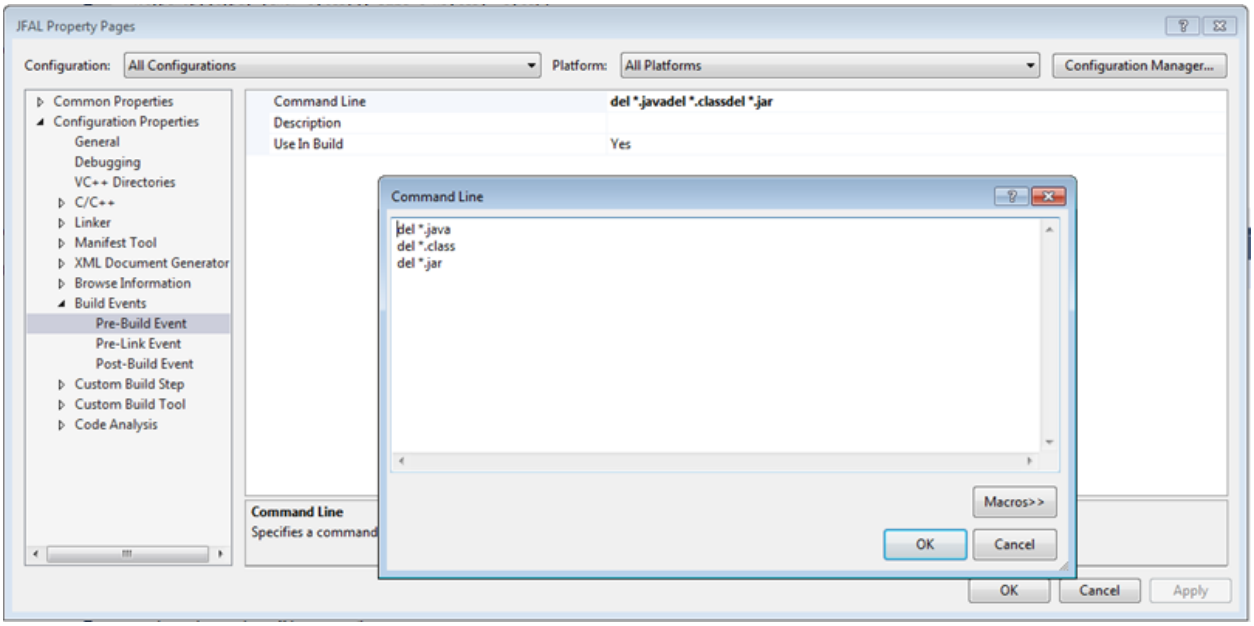
The JFAL project is more complex. This project builds a JNI layer over the EFAL API and then uses that JNI layer to implement the JFALShell application. This project depends on SWIG and also depends on a JAVA SDK to be preinstalled.

The project consists of the following source files:

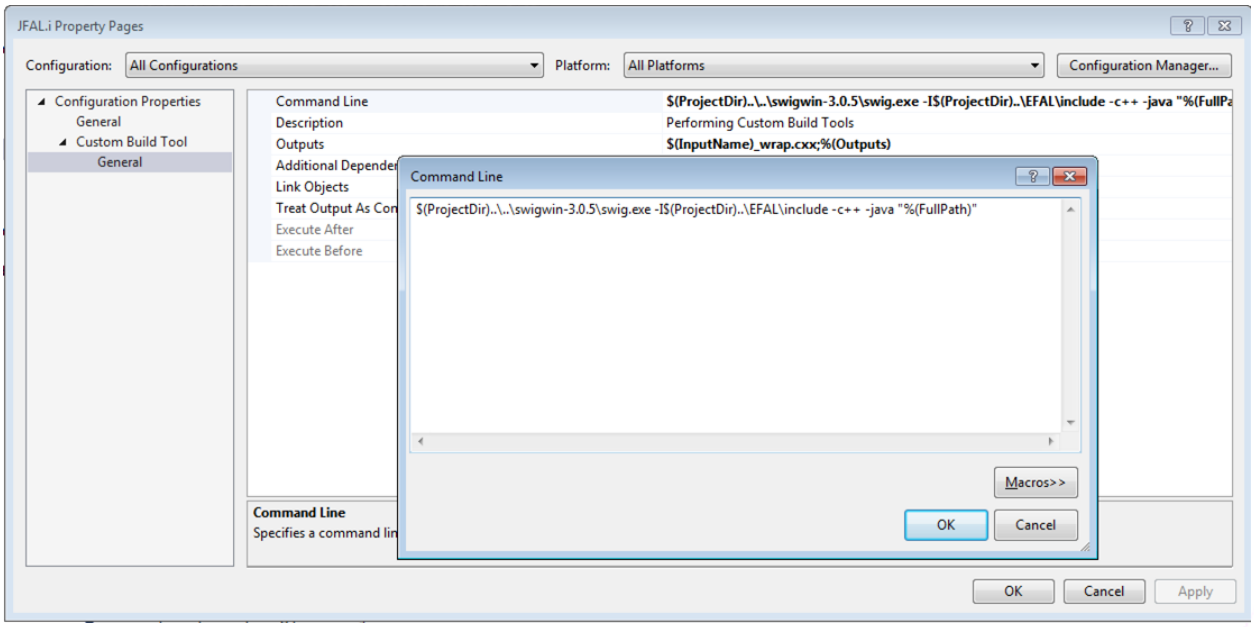
- JFAL.i – this is the main input file for SWIG. It includes the EFAL.h header file and transforms it into a JNI layer. It generates the C++ file named JFAL_wrap.cxx which is used by the project to build the native portion of the JNI layer.
- Wchar.i – utility interface declarations used by the SWIG process.
- JFAL_wrap.cxx – generated by the SWIG process. The project primarily will compile this into JFAL.dll.
- JFALShell.java

[1] The Solution folder also contains a copy of SWIG (http://www.swig.org/) used for generating the JNI layer for Java integration. The final version of the EFAL SDK will most likely not include a copy of SWIG but it is provided for simplicity in this pre-Beta package.

The project has several parts of the build. First, at the project level, there is a pre-build event that will delete any intermediate JAVA files (*.java, *.class, and *.jar) left over from a previous build as the following dialog shows



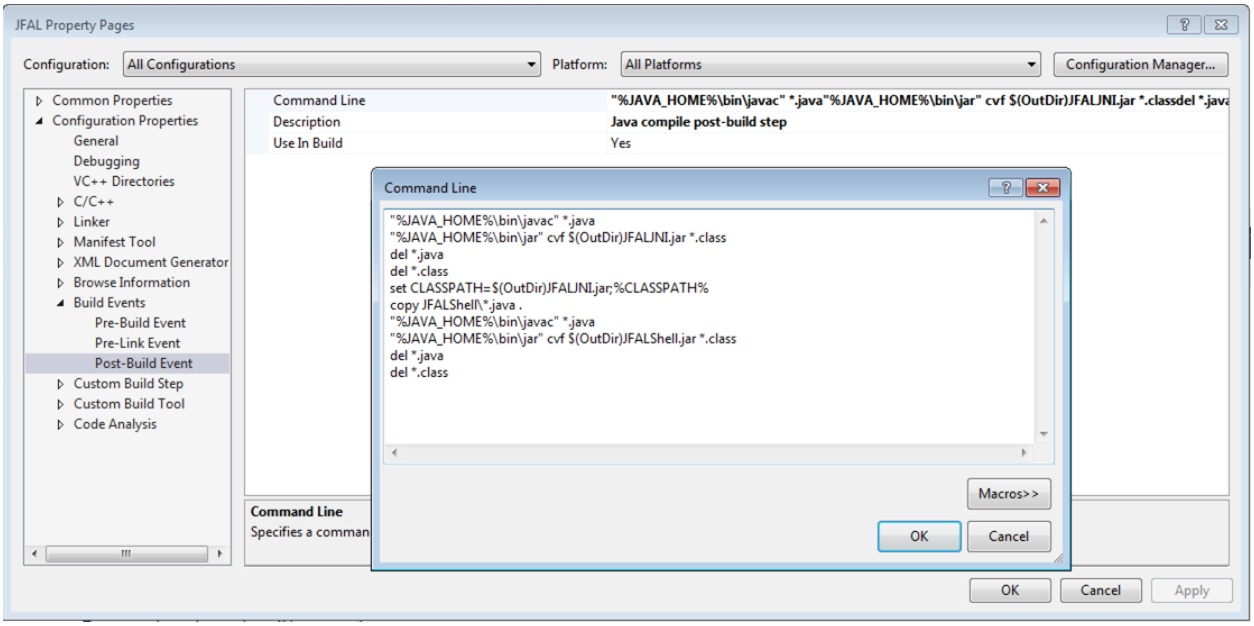
Next, the SWIG processing is executed as a custom build tool defined on the JFAL.i file as the following dialog illustrates.[2]



[2] Note that the command is specified as `$(ProjectDir)..\..\swigwin-3.0.5\swig.exe` referencing the copy of SWIG supplied with this distribution. This will most likely change in the future to have a dependency on the system PATH.

The project then builds the JFAL_wrap.cxx file and links it to the EFAL.lib (for the appropriate architecture) into JFAL.dll.

The final step of the project build is to combine the JFAL source JAVA generated from SWIG into a JAR file and then compile the JFALShell source into a separate JAR file. The project has a Post-Build event defined as the following dialog illustrates:



Xport Project

The Xport project contains some utility scripts for finishing up the build process. The custom build tool for this project executes a script file named xport.bat. This script copies the build artefacts from the efalshell, nfal, nfalshell, and jfal projects into the appropriate folder containing the SDK (export folder). The project also contains some other utility sample files such as a utility BAT file for launching JFALShell and a sample input file for use in EFALShell, NFALShell, or JFALShell console applications that demonstrates the commands using the sample data provided. These utility scripts will also be copied into the export directory by xport.bat.

Sample Data and script file

The SDK distribution contains a folder named Data which contains sample files in MapInfo (Native), MapInfo Extended (NativeX) and OGC GeoPackage formats. The solution project (Xport) also contains a script file named sample_command.txt that is exported along with the sample applications.

The sample commands file contains the following contents

```
# ------------------------------------------------------------
# Sample EFALShell (or JFALShell) command file
# ------------------------------------------------------------
# This script will open a series of tables and issue several
# commands that will demonstrate how to use EFALShell commands.
# The sample data is provided in 3 formats: MapInfo TAB (Native),
# MapInfo Enhanced (NativeX), and OGC GeoPackage.
# ------------------------------------------------------------
' open the source tables
open ..\..\data\nativex\USA.TAB
open ..\..\data\nativex\USA_CAPS.TAB
open ..\..\data\nativex\USCTY_1K.TAB
open ..\..\data\nativex\US_CNTY.TAB
open ..\..\data\nativex\US_HIWAY.TAB
'
' List the catalog of opened tables and then display the metadata
' for each
tables
desc USA
desc USA_CAPS
desc USCTY_1K
desc US_CNTY
desc US_HIWAY
'
close US_HIWAY
tables
'
' Issue a few queries against the tables
'
SELECT count(*) FROM USA
SELECT STATE, Count(*) FROM US_CNTY GROUP BY STATE
SELECT STATE, Count(*) FROM US_CNTY GROUP BY STATE ORDER BY 2
SELECT STATE, POP_1990 FROM USA_CAPS WHERE POP_1990 > 500000 ORDER BY POP_1990
'
' Join Query
'
SELECT A.City as City, B.State_Name as StateName FROM USCTY_1K as A, USA as B WHERE A.STATE = B.STATE and
B.STATE_NAME = 'NEW YORK'
'
' Demonstrate use of variables in SQL statements
'
set @pop=500000
show @pop
show
SELECT STATE, POP_1990 FROM USA_CAPS WHERE POP_1990 > @pop ORDER BY POP_1990
set @NY=SELECT OBJ FROM USA WHERE STATE = 'NY'
show @NY
SELECT City FROM USCTY_1K WHERE Obj within @NY
'
' Save a copy of USA (we will use it to demonstrate INSERT, UPDATE, and DELETE commands)
'
Save table USA in TEMPUSA.tab
desc TEMPUSA
SELECT Count(*) FROM TEMPUSA
'
DELETE FROM TEMPUSA WHERE STATE = 'FL'
SELECT Count(*) FROM TEMPUSA
```

```
'
SELECT STATE, STATE_NAME FROM TEMPUSA WHERE STATE = 'NY'
UPDATE TEMPUSA SET State_Name = 'New York' WHERE STATE = 'NY'
SELECT STATE, STATE_NAME FROM TEMPUSA WHERE STATE = 'NY'
'
INSERT INTO TEMPUSA (STATE, STATE_NAME, OBJ) VALUES ('FB', 'Foo Bar', @NY)
SELECT Count(*) FROM TEMPUSA
select * from TEMPUSA Where STATE='FB'
```

To run the sample application after successfully building the Visual Studio solution, open a command prompt, set the current directory to the "Export\ux64" or "Export\uw32" folder depending on the architecture that was built.

To run EFALShell, simply execute EFALShell. The sample commands can be copy/pasted one-by-one at the prompt or can run as a batch with the command

```
EFALShell < sample_commands.txt
```

Executing NFALShell is identical to EFALShell.

Executing JFALShell requires a JDK present for the desired architecture (64 or 32 bit). The Xports project contains a script file to simplify running the JFALShell application. It is defined as

```
@echo off
cls
"%JAVA_HOME%\bin\java.exe" -classpath JFAL.jar;JFALJNI.jar;JFALShell.jar JFALShell
```

This script simply adds the three JAR files that the solution created to the class path and runs Java.exe with JFALShell as the main entry point.