Line Item Explanation Application
DataAccess Namespace
Class and Method Components

Document Status (Draft)

Document Completion Date (08/2003)

Version 1.0

SPM ID #: 10238A0
CID #: 381354

Copyrights and Trademarks

Document Revision History

Date	Author	Version	Document Changes and Reasons
7/16/2003	Lance Smith	.8	Initial Draft
8/01/2003	Lance Smith	.95	Consistency Items
8/13/2003	Lance Smith	1.0	`Explanations.cs` (`SaveExplanationsDatasetAdd`) transaction rollback if updates to the `patterns`, `explanations`, or `patterns_explanations` table fail `OccurencesDAC.cs` (`InsertOccurrences`) Insert/update records in `line_items`, `context`, `content`, and `occurences` database tables Reconciles `ExplanationsDataSet` with database Add transaction rollback if updates failed

Document Scope

This document provides detailed design information about the **DataAccess** class and its subordinate methods, structures, data access queries, .ASP pages, .XML files, .XSL files, and components used in the Verizon.com **Line Item Explanation** application.

Notation Used

Notation	Meaning within the Use Case
`CustomerTable`	Classes and Method Names Pascal naming convention with each word's first letter capitalized.
`customerTable`	Internal variables and structures Camelback notation – Pascal naming with the first letter as lower-case.
`statement_id` or `TABLE_INSERT_`	Database columns, record, or table names with underscore characters Use with SQL Queries

Table of Contents

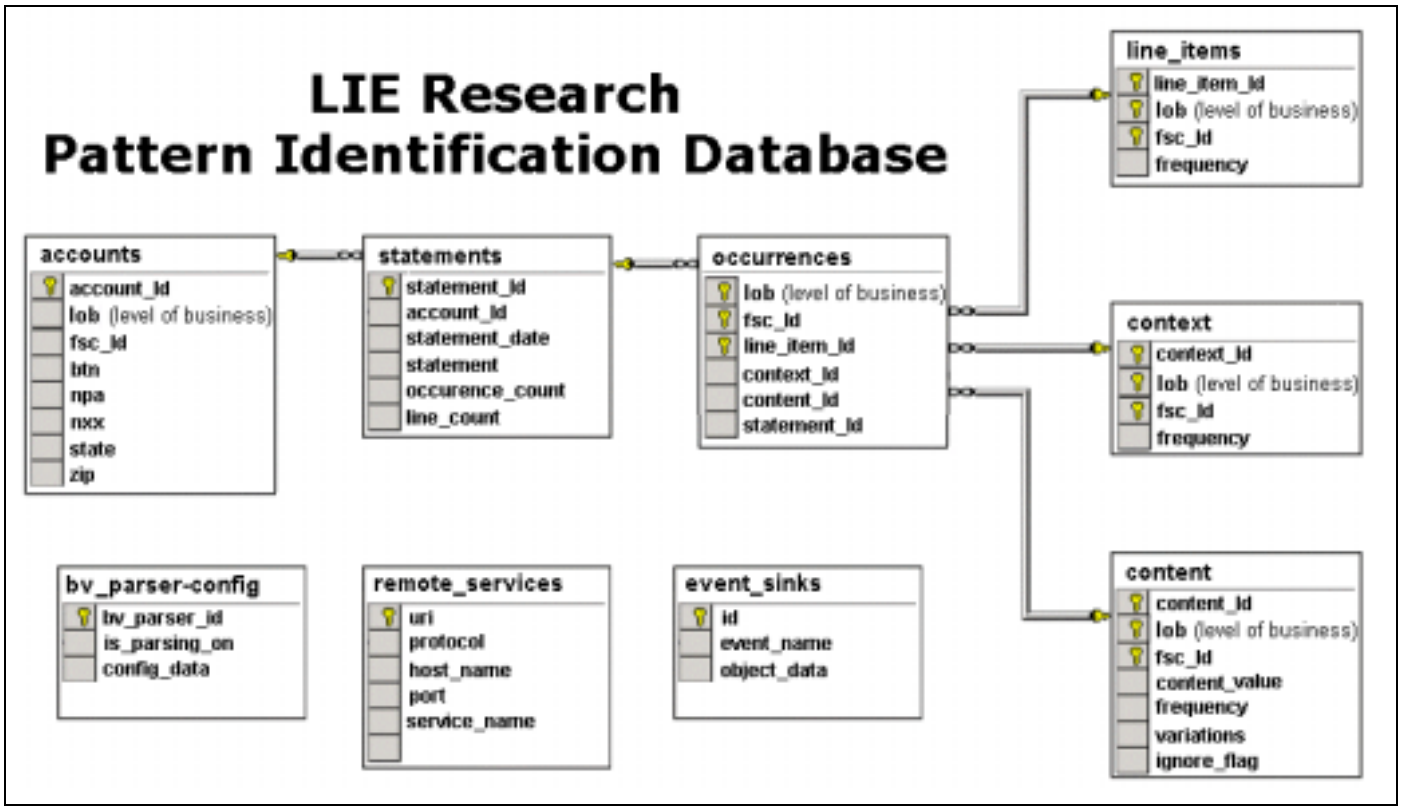
DataAccess Namespace Overview

Introduction

Web Services and the interactive LIE Administrative tool (using ASP.NET) use the DataAccess Namespace classes to manipulate explanations and patterns within the LIE Production database. The Statement class is used to load a group of Jurisdiction's statements into the LIE Pattern Research database and parse the statements for appropriate patterns that the CMS content teams will link to billing statement explanations.

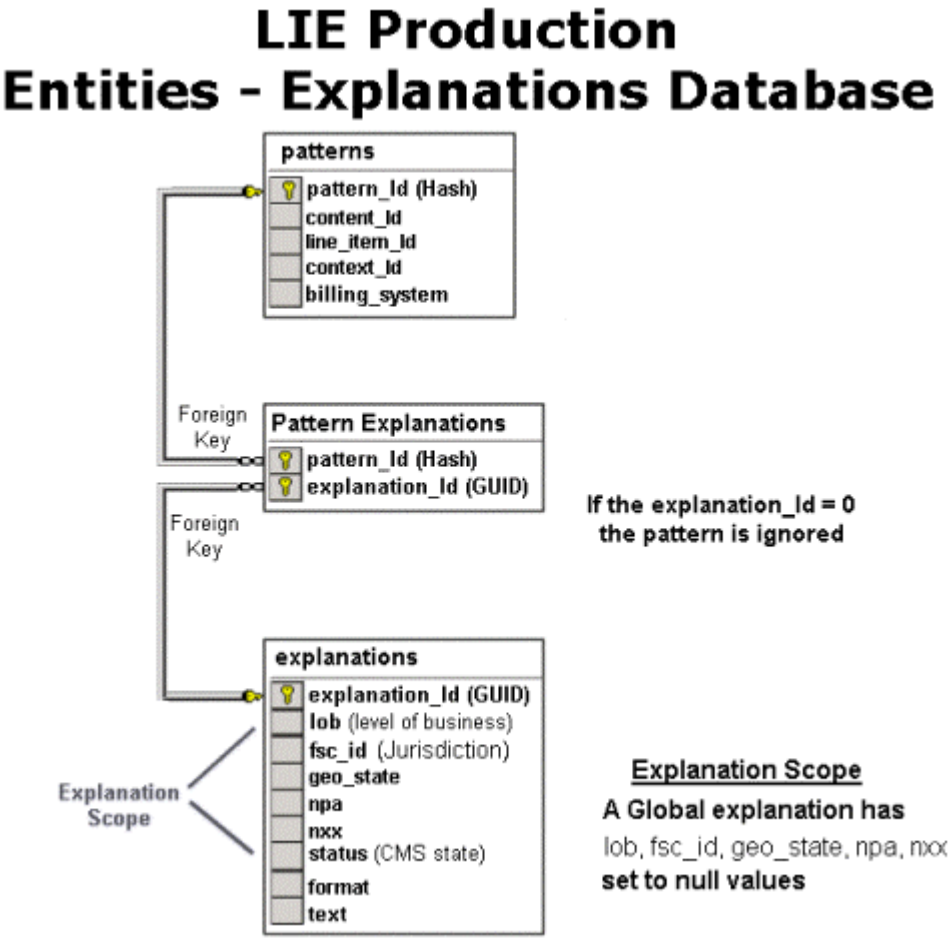
LIE Research Pattern Identification Database Schema

This graphic describes the table elements found in the LIE research database:



LIE Production Database Schema

This graphic describes the table elements found in the high-performance portion of the LIE database:



Class Index

Class	Description
EventsSinkDAC	Facilitates data access to the `event_sinks` table.
Explanations	Provides access to the Explanations and Patterns tables of LIE production database.
OccurencesDAC	Inserts and updates records in the LIE Research Pattern Identification database tables: `statements`, `occurrences`, `line_items`, `context`, and `content`.
Patterns	Provides data access operations for the `patterns` table of the LIE production database.
RemoteServicesDAC	Facilitates data access to the `remote_services` LIE database table.
Statement	Provides access to the LIE and Oracle data sources to retrieve and store billing statements.

Method Index

Method	Purpose
AddMapping	Add a row to the `patterns_explanations` table that permits the many patterns to many explanations mapping
CleanExplanationsTables	Execute the stored procedure to clean all LIE Production database tables: `patterns, explanations, patterns_explanations`
DeleteExplanation	DELETEs a single record from the LIE `explanations` table.
DeleteMapping	Permits LIE to remove the relationship between a bill statement line item and an explanation.
DeletePattern	Deletes pattern row from the LIE production `patterns` table
DeleteStatement	Delete a statement record from the LIE research `statements` and `occurrences` table
Dispose	Dispose of the pattern object resources found in `SqlDataAdapter`
Dispose `(overloaded)`	Dispose of a SELECT command instance
EventSinksDAC	Build SQL commands to insert, update, delete, and select records from the `event_sinks` table in the LIE database.
Explanations	Default Constructor for `Explanations` class.
GetAccount	Get customer account information from the BillView Oracle database
GetAllServices	Read all information in the LIE production `remote_services` table into an in-memory dataset
GetConnection	Constructs actual data source connection using the LIE production database connection string passed from the configuration file.
GetDeleteExplanationCommand	Initializes and returns parameterized DELETE `SqlCommand` object for the LIE `explanations` table.
GetExplanationsDataset	Returns `ExplanationsDataSet` with all records from the database
GetExplanationIdForPattern	For a given statement content identifier, line item identifier, and context identifier get the explanation identifier (GUID) from the LIE Production `patterns` table.
GetExplanationTextById	Retrieve the portion of the explanation object stored in the CMS repository.
GetFromBackend	Retrieves billing statement from backend using BillViewObjectR2 through INTEROP.
GetFromLieDb	Retrieves most recent billing statement from LIE database for this account number and jurisdiction.
GetFromLieDb `(overloaded)`	Retrieves billing statement from LIE database for this account number, statement date, and jurisdiction.
GetFromLieDb `(Multistatement overloaded)`	Retrieves the desired number of billing statements from LIE database for this jurisdiction.
GetInsertExplanationCommand	Initializes and returns parameterized INSERT `SqlCommand` object for the LIE `explanations` table.

Method	Purpose
GetNumberOfOccurrences	Returns number of occurrences in the LIE research `statements` table for this statement ID (if it was decomposed)
GetOccurencesDataset [**Obsolete**]	For a given Level of Business and Jurisdiction (Reports) copy rows from from the LIE research pattern identification database tables: `occurrences, line_items, content, context`
GetSelectCommand	Returns a SELECT command that uses the explanation Id as a unique key.
GetUpdateExplanationCommand(	Initializes and returns parameterized UPDATE `SqlCommand` object for the LIE `explanations` table.
InitializeComponent	Sets-up the INSERT and UPDATE SQL Server commands for the LIE Research Pattern Identification Research database tables: `occurrences, line_items, context, and content`.
InsertAccount	Inserts a single record into the LIE research pattern identification `accounts` table.
InsertExplanation	INSERTs a single record into the LIE `explanations` table.
InsertPattern	Inserts new pattern row into the LIE production `patterns` table
InsertOccurrences	Executes the INSERT and UPDATE queries for the LIE Research Pattern Identification Research database tables: `occurrences, line_items, context, and content`.
InsertTable	Saves entire table content to the LIE database
IsExistingAccount	Checks if the customer account exists in the LIE research pattern identification database table `accounts`
Patterns	Patterns class constructor that determines the data source connection string and creates a new instance of the `SqlDataAdapter`
RemoteServicesDAC	Initialize `cmdSelectAllServices` command
SaveExplanationsDataset	Save all explanations received from CMS synchronization.
SaveToLieDb	Saves XML statement in the `statements` table of the LIE database
Statement `(overloaded)`	Get data source connection strings for the LIE Production and BillView Production data sources from the `Web.config` file.
UpdateExplanation	UPDATEs a single record into the LIE `explanations` table.
UpdateMapping	Updates the `explanation_id` for a given `pattern_id`. Permits LIE to revise the explanation(s) associated with a billing statement line item
UpdateStatements	Updates `occurrence_count` and `line_count` fields in the LIE research `statements` table for a given `statement_id`

EventsSinkDAC

Summary

Facilitates data access to the `event_sinks` table. This code is currently in development so that we can remotely control the LIE production application that interacts with BillView.

Methods

```
public EventSinksDAC()
```

Properties

Name	Type	Description

Enumerations or Constants

None

Data Members

None

Significant .NET NameSpaces Used

System.Data	Build components that efficiently manage data from multiple data sources.
System.Data.Common	Collection of classes used by a .NET data provider to access a data source (database) in the managed space.
System.Data.SqlClient	Collection of classes used by MS SQL Server to access a data source in the managed space.
System.Data.SqlTypes	Native data types within SQL Server that seeks to prevent type conversion errors.

Configuration Constants

```
SQL Connection String - CBTXDB01;initial catalog=LIE;password=password1;persist
security info" +"=True;user id=lie_dev;workstation id=PPETROV;packet size=4096"
```

EventSinksDAC method

LIE.DataAccess Namespace **- EventsSinkDAC** Class **–** public static **EventSinksDAC** Method

Purpose	Build SQL commands to insert, update, delete, and select records from the `event_sinks` table in the LIE database.
Description	Use `SqlDataAdapter` class methods to build connection strings and commands

Synopsis	`public EventSinksDAC()`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

CBTXDB01;initial catalog=LIE;password=password1;persist security info" +
"=True;user id=lie_dev;workstation id=PPETROV;packet size=4096"

Explanations

Summary

Provides access to the Explanations table of LIE production database from the `ExplanationFlow` classes and the Web Services methods. Several methods construct the SQL query that is used by another class method to execute the actual database operation (INSERT, UPDATE, or DELETE).

Methods

`public Explanations()` – **Default Constructor**

`private SqlConnection GetConnection()`

`private SqlCommand GetInsertExplanationCommand(ExplanationData explanation)`

`private SqlCommand GetUpdateExplanationCommand(ExplanationData explanation)`

`private SqlCommand GetDeleteExplanationCommand(string explanationId)`

`public bool InsertExplanation(ExplanationData explanation)`

`public bool UpdateExplanation(ExplanationData explanation)`

`public bool DeleteExplanation(string explanationId)`

<<Commented Out>> `public PatternData GetPattern(string contentId, string lineItemId, string contextId)`

`public string GetExplanationIdForPattern(string contentId, string lineItemId, string contextId)`

`public object GetExplanationTextById(string explanationId)`

<<Commented Out>> `public ExplanationData GetExplanation(string explanationId)`

`public void CleanExplanationsTables()`

`public void SaveExplanationsDataset(ExplanationsDataSet explanations)`

<<Not in Production>> `public ExplanationsDataSet GetExplanationsDataset()`

Properties

Name	Type	Description

Enumerations

```
EXPLANATIONS_TABLE_NAME = "explanations"
```

Field names used in constructed queries:

```
EXPLANATION_ID_PARAM = "@explanation_id"
LOB_PARAM = "@lob"
FSC_ID_PARAM = "@fsc_id"
GEO_STATE_PARAM = "@geo_state"
NPA_PARAM = "@npa"
NXX_PARAM = "@nxx"
```

Data Members

```
Explanation _instance – Class to establish the SQL connection string
```

Significant .NET NameSpaces Used

`System.Collections`	Interfaces and classes that define collections of objects, such as lists, queues, bit arrays, hash tables and dictionaries.
System.Collections.Specialized	Specialized and strongly-typed collections; for example, a linked list dictionary, a bit vector, and collections that contain only strings.
`System.Configuration`	Access .NET Framework configuration settings and handle errors in configuration (*.config) files .
`System.Data`	Build components that efficiently manage data from multiple data sources. In a disconnected scenario (such as the Internet), ADO.NET provides the tools to request, update, and reconcile data in multiple-tier systems.
`System.Data.Common`	Collection of classes used by a .NET data provider to access a data source (database) in the managed space.
`System.Data.SqlClient`	Collection of classes used by MS SQL Server to access a data source in the managed space.
`LIE.Data`	Generated C# code that defines the objects mapped into relational records.

Configuration Constants

```
lieDbConnString
```

Explanations method

LIE.DataAccess Namespace **- Explanations** Class **–** public **Explanations** Method

Modifier: `public`

Purpose	Default Constructor for Explanations class.
Description	Determines the LIE database connection string for the data source Creates new instance of the `SQLDataAdapter`
Synopsis	`public Explanations()`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

Uses configuration parameter for LIE database: `lieDbConnString`

Default Connection String: `"Network Library=DBMSSOCN;Data Source=cbtxdb01,1433;Initial Catalog=LIE;Integrated Security=false;User ID=lie_dev;Password=password1;"`

GetConnection method

LIE.DataAccess Namespace **- Explanations** Class **–** private **GetConnection** Method

Purpose	Constructs the actual data source connection using the LIE production database connection string passed from the configuration file.
Description	

Synopsis	`private SqlConnection GetConnection()`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

GetInsertExplanationCommand method

LIE.DataAccess Namespace **- Explanations** Class **–** private **GetInsertExplanationCommand** Method

Purpose	Initializes and returns parameterized INSERT `SqlCommand` object for the LIE `explanations` table.	
Description	1. Create INSERT command object (with parameters) and associated connection if not created already 2. Populate the command with the explanation object values 3. Return it as a sqlCommand object	
Synopsis	`private SqlCommand GetInsertExplanationCommand(ExplanationData explanation)`	
Parameters	`explanation`- [ExplanationData]	Explanation object to INSERT into the LIE database
Returns	`SqlCommand`	Resulting INSERT SQL command object
Exceptions Thrown	*None*	

Related Information

```
"INSERT INTO explanations (explanation_id, lob, fsc_id, geo_state,
text, format) VALUES(@explanation_id, @explanation_text, @format)";
```

GetUpdateExplanationCommand method

LIE.DataAccess Namespace **- Explanations** Class **–** private **GetUpdateExplanationCommand** Method

Purpose	Initializes and returns parameterized UPDATE `SqlCommand` object for the LIE explanations table.	
Description	1. Create UPDATE command object (with parameters) and associated connection if not created already 2. Populate the command with the explanation object values 3. Return it as a sqlCommand object	
Synopsis	`private SqlCommand GetUpdateExplanationCommand(ExplanationData explanation)`	
Parameters	`explanation`- [ExplanationData]	Explanation object to UPDATE in the LIE database
Returns	`SqlCommand`	Resulting UPDATE SQL command object
Exceptions Thrown	*None*	

Related Information

```
"UPDATE explanations SET lob=@lob, fsc_id=@fsc_id,
geo_state=@geo_state, npa=@npa, nxx=@nxx, status=@status,
format=@format, text=@text WHERE explanation_id=@explanation_id"
```

GetDeleteExplanationCommand method

LIE.DataAccess Namespace **- Explanations** Class **–** private **GetDeleteExplanationCommand** Method

Purpose	Initializes and returns parameterizedDELETE `SqlCommand` object for the LIE explanations table.	
Description	1. Create DELETE command object (with parameters) and associated connection if not created already 2. Populate the command with the explanation object values 3. Return it as a sqlCommand object	
Synopsis	`private SqlCommand GetDeleteExplanationCommand(ExplanationData explanation)`	
Parameters	`explanation`-[ExplanationData]	Explanation object to DELETE into the LIE database
Returns	`SqlCommand`	Resulting DELETE SQL command object
Exceptions Thrown	*None*	

Related Information

```
"DELETE FROM explanations WHERE explanation_id=@explanation_id"
```

InsertExplanation method

LIE.DataAccess Namespace **- Explanations** Class **–** public **InsertExplanation** Method

Purpose	INSERTs a single record into the LIE explanations table.	
Description	1. Call the private method to build the query if needed 2. Execute the INSERT query 3. Test the number of affected rows is nonzero	
Synopsis	`public bool InsertExplanation(ExplanationData explanation)`	
Parameters	`explanation`-[ExplanationData]	Explanation object to INSERT into the LIE database
Returns	True	Explanation successfully inserted into the LIE database
Exceptions Thrown	*None*	

Related Information

`private GetInsertExplanationCommand(explanation)`

UpdateExplanation method

LIE.DataAccess Namespace **- Explanations** Class **–** public **UpdateExplanation** Method

Purpose	UPDATEs a single record into the LIE explanations table.	
Description	1. Call the private method to build the query if needed 2. Execute the UPDATE query 3. Test the number of affected rows is nonzero	
Synopsis	`public bool InsertExplanation(ExplanationData explanation)`	
Parameters	`explanation`-[ExplanationData]	Explanation object to UPDATE in the LIE database
Returns	True	Explanation successfully UPDATED in the LIE database
Exceptions Thrown	*None*	

Related Information

`private GetUpdateExplanationCommand(explanation)`

DeleteExplanation method

LIE.DataAccess Namespace **- Explanations** Class **–** public **DeleteExplanation** Method

Purpose	DELETEs a single record from the LIE explanations table.	
Description	1. Call the private method to build the query if needed 2. Execute the DELETE query 3. Test the number of affected table rows is nonzero	
Synopsis	`public bool DeleteExplanation(ExplanationData explanation)`	
Parameters	`explanation`-[ExplanationData]	Explanation object to DELETE from the LIE database
Returns	True	Explanation successfully DELETE from the LIE database
Exceptions Thrown	*None*	

Related Information

`private GetDeleteExplanationCommand(explanation)`

GetExplanationIdForPattern method

LIE.DataAccess Namespace **- Explanations** Class **–** public static **GetExplanationIdForPattern** Method

Purpose	For a given statement content identifier, line item identifier, and context identifier get the explanation identifier (GUID) from the LIE Production **pattern** table.	
Description	1. Execute the SELECT query shown below 2. Return the explanation identifier as a string	
Synopsis	`public string GetExplanationIdForPattern(string contentId, string lineItemId, string contextId)`	
Parameters	`contentId`-[string]	Content Identifier (hash string created from the actual text in the statement)
	`lineItemId`-[string]	Line Item Identifier – actual text from the bill that was normalized
	`contextId`-[string]	Context Identifier – actual text that corresponds to the node that identifies this line item in the bill XML.
Returns	String	Explanation Identifier obtained from the pattern LIE table using the three input strings
Exceptions Thrown	*None*	

Related Information

SELECT explanation_id FROM patterns WHERE (content_id='{0}') AND (line_item_id='{1}') AND (context_id='{2}')", contentId, lineItemId, contextId);

GetExplanationTextById method

LIE.DataAccess Namespace **- Explanations** Class **–** public **GetExplanationTextById** Method

Purpose	Retrieve the portion of the explanation object stored in the CMS repository.	
Description	1. Format the SELECT query shown below in a string 2. Use the LIE database data source connection 3. Return the text portion of the explanation object	
Synopsis	`public object GetExplanationTextById(string explanationId)`	
Parameters	explanationId-[string]	Explanation identifier (GUID)
Returns	object	Text of the explanation that was created in CMS
Exceptions Thrown	*None*	<<Description>>

Related Information

```
"SELECT text FROM explanations WHERE explanation_id='{0}'",
explanationId)
```

CleanExplanationsTables method

LIE.DataAccess Namespace **- Explanations** Class **–** public **CleanExplanationsTables** Method

Purpose	Execute the stored procedure to clean all LIE Production database tables. `patterns, explanations, patterns_explanations`
Description	1. Use the LIE data source connection string 2. Execute the stored procedure `CleanExplanationsTables`
Synopsis	`public void CleanExplanationsTables()`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

SaveExplanationsDataset method

LIE.DataAccess Namespace **- Explanations** Class **–** public **SaveExplanationsDataset** Method

Purpose	Save all explanations received from CMS synchronization.	
Description	1. Delete all content from the LIE Production database tables (**CleanExplanationsTables**) 2. Use the LIE production data source connection 3. Select all the explanations received.	
Synopsis	`public void SaveExplanationsDataset(ExplanationsDataSet explanations)`	
Parameters	explanations- [ExplanationsDataSet]	In-cache memory dataset of explanations received from the CMS team.
Returns	*None*	
Exceptions Thrown	*None*	

Related Information

GetExplanationsDataset method

LIE.DataAccess Namespace **- Explanations** Class **–** public **`GetExplanationsDataset`** Method

Purpose	Returns `ExplanationsDataSet` with all records from the database	
Description		
Synopsis	`public ExplanationsDataSet GetExplanationsDataset()`	
Parameters	*None*	
Returns	`ExplanationsDataSet`	Object holding all LIE production database information
Exceptions Thrown	*None*	

Related Information

None

OccurrencesDAC

Summary

Inserts and updates records in the LIE Research Pattern Identification Research database tables: `statements, occurrences, line_items, context, and content.`


```
private void InitializeComponent()
public void InsertOccurrences(DataSet ds)
```

Properties

Name	Type	Description

Enumerations

None

Data Members

None

Significant Namespaces Used

Configuration Constants

None

InitializeComponent method

LIE.DataAccess Namespace **- OccurrencesDAC** Class **–** private **InitializeComponent** Method

Purpose	Sets-up the INSERT and UPDATE SQL Server commands for the LIE Research Pattern Identification Research database tables: `occurrences, line_items, context, and content`.
Description	
Synopsis	`private void InitializeComponent()`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

InsertOccurrences method

LIE.DataAccess Namespace **- OccurrencesDAC** Class **–** public **InsertOccurrences** Method

Purpose	Executes the INSERT and UPDATE queries for the LIE Research Pattern Identification Research database tables: `occurrences, line_items, context, and content`.
Description	1. Update `line_tems` table records with INSERT and UPDATE queries 2. Update `context` table records with INSERT and UPDATE queries 3. Update `content` table records with INSERT and UPDATE queries 4. Update `occurences` table records with INSERT and UPDATE queries
Synopsis	`public void InsertOccurrences(DataSet ds)`
Parameters	`ds`-[DataSet]
Returns	*None*
Exceptions Thrown	*ApplicationException* `Failed to save ExplanationsDataset`

Related Information

Patterns

Summary

Provides access to the Patterns table of LIE production database from the `ExplanationFlow` classes and the Web Services methods. Several methods construct the SQL query that is used by another class method to execute the actual database operation (INSERT, UPDATE, or DELETE).

`public Patterns()` – Establish Data source connection string and adpter

`public void Dispose()` – Dispose of Patterns object resources

```
protected virtual void Dispose(bool disposing)
private SqlCommand GetSelectCommand()
public bool InsertPattern(PatternData pattern)
public bool DeletePattern(PatternData pattern)
public bool AddMapping(string patternId, Guid explanationId)
public bool UpdateMapping(string patternId, Guid explanationId)
public bool DeleteMapping(string patternId, Guid explanationId)
```

Properties

Name	Type	Description

Enumerations

```
string PATTERNS_TABLE = "patterns";
```

Field names used in constructed queries:

```
string PATTERN_ID_PARAM = "@line_item_id"
string CONTENT_ID_PARAM = "@content_id"
string LINE_ITEM_ID_PARAM = "@line_item_id"
string CONTEXT_ID_PARAM = "@context_id"
string BILLING_SYSTEM_PARAM = "@content_id"
string EXPLANATION_ID_PARAM = "@explanation_id"
```

Data Members

```
Instance of itself
```

Significant .NET NameSpaces Used

`System.ComponentModel`	Run-time and design-time behavior of components and controls. Namespace includes the base classes/interfaces for implementing

	attributes and type converters, binding to data sources, and licensing components.
`System.Configuration`	Access .NET Framework configuration settings and handle errors with configuration (*.config) files .
`System.Data`	Build components that efficiently manage data from multiple data sources. Provides the tools to request, update, and reconcile data in multiple-tier systems.
`System.Data.Common`	Collection of classes used by a .NET data provider to access a data source (database) in the managed space.
`System.Data.SqlClient`	Collection of classes used by MS SQL Server to access a data source in the managed space.
`LIE.Data`	Generated C# code that defines the objects mapped into relational records.

Configuration Constants

`SqlDataAdapter` - Represents a set of data commands and a database connection that are used to fill the DataSet and update a SQL Server database. This class cannot be inherited.

```
selectQuery = "SELECT * FROM patterns WHERE (pattern_id IN " +"(SELECT pattern_id
FROM patterns_explanations " +"WHERE explanation_id = @explanation_id))";
```

Patterns method

LIE.DataAccess Namespace **- Patterns** Class **–** public **patterns** Method

Purpose	Patterns class constructor that determines the data source connection string and creates a new instance of the `SqlDataAdapter`
Description	1. Read the contents of the `lieDbConnString` configuration parameter 2. Use default value if configuration service returns nothing 3. Construct a new **SqlDataAdapter** that bridges between the source and application by mapping `Fill` which changes the data in the **DataSet** to match the data in the data source, and `Update`, which changes the data in the data source to match the data in the **DataSet**.
Synopsis	`public Patterns()`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

```
Default LIE database connection string:
"Network Library=DBMSSOCN;Data Source=cbtxdb01,1433;Initial
Catalog=LIE;Integrated Security=false;User
ID=lie_dev;Password=password1;"
```

Dispose method (derived)

LIE.DataAccess Namespace **- Patterns** Class **–** public **Dispose** Method

Purpose	Dispose of the pattern object resources found in `SqlDataAdapter`
Description	
Synopsis	`public void Dispose()`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

Dispose method (virtual)

LIE.DataAccess Namespace **- Patterns** Class **–** protected **Dispose** Method

Purpose	Dispose of a SELECT command instance	
Description	1. Check to be sure this method is not already disposing of the pattern 2. If the data adapter and the SELECT command are not null, dispose of the SELECT command connection 3. Finally dispose of the `SqlDataAdapter` object.	
Synopsis	`protected virtual void Dispose(bool disposing)`	
Parameters	disposing-[bool]	Boolean flag set during disposal of a pattern instance
Returns	*None*	
Exceptions Thrown	*None*	

Related Information

GetSelectCommand method

LIE.DataAccess Namespace **- Patterns** Class **–** private **`GetSelectCommand`** Method

Purpose	Returns a SELECT command that uses the explanation Id as a unique key	
Description	Build the SELECT command if an instance does not exist	
Synopsis	`private SqlCommand GetSelectCommand()`	
Parameters	*None*	
Returns	`SqlCommand`	Returns SELECT SQL command with the explanation Id as a unique key
Exceptions Thrown	*None*	

Related Information

InsertPattern method

LIE.DataAccess Namespace **- Patterns** Class **–** public static **InsertPattern** Method

Purpose	Inserts new pattern row into the LIE production `patterns` table	
Description	1. Use the LIE data source connection string from the constructor 2. Execute the query and test for success	
Synopsis	`public bool InsertPattern(PatternData pattern)`	
Parameters	pattern-[PatternData]	Pattern containing information to insert into the LIE database
Returns	True	Row successfully inserted into the `patterns` table
Exceptions Thrown	*None*	

Related Information

```
INSERT INTO patterns VALUES('{0}','{1}','{2}','{3}','{4}')",
pattern.Id, pattern.ContentId, pattern.LineItemId,
pattern.ContextId, pattern.BillingSystem
```

`BillingSystem` is the jurisdiction integer of the billing statement

DeletePattern method

LIE.DataAccess Namespace **- Patterns** Class **–** public **DeletePattern** Method

Purpose	Deletes pattern row from the LIE production `patterns` table	
Description	1. Use the LIE data source connection string from the constructor 2. Execute the query and test for success	
Synopsis	`public bool DeletePattern(PatternData pattern)`	
Parameters	pattern-[PatternData]	Pattern containing the content, line item, context, and jurisdiction identifiers used to remove the row.
Returns	Boolean	True - Row successfully deleted from the `patterns` table
Exceptions Thrown	*None*	

Related Information

```
"DELETE FROM patterns WHERE (content_id='{0}') AND
(line_item_id='{1}') AND (context_id='{2}') AND
(billing_system='{3}')", pattern.ContentId, pattern.LineItemId,
pattern.ContextId, pattern.BillingSystem)
```

`BillingSystem` is the jurisdiction integer of the billing statement

AddMapping method

LIE.DataAccess Namespace **- Patterns** Class **–** public **AddMapping** Method

Purpose	Add a row to the `patterns_explanations` table that permits the many patterns to many explanations mapping.	
Description	1. Use the LIE data source connection string from the constructor 2. Execute the query and test for success	
Synopsis	`public bool AddMapping(string patternId, Guid explanationId)`	
Parameters	`patternId`-[`string`]	Pattern Identifier created using a hash function
	`explanationId`-[`Guid`]	GUID assigned by the CMS repository software
Returns	Boolean	True – row successfully added to the LIE `patterns_explanations` table
Exceptions Thrown	*None*	

Related Information

```
"INSERT INTO patterns_explanations VALUES('{0}', '{1}')", patternId,
explanationId)
```

UpdateMapping method

LIE.DataAccess Namespace **- Patterns** Class **–** public **UpdateMapping** Method

Purpose	Updates the `explanation_id` for a given `pattern_id`. Permits LIE to revise the explanation(s) associated with a billing statement line item.	
Description	1. Use the LIE data source connection string from the constructor 2. Execute the query and test for success	
Synopsis	`public bool UpdateMapping(string patternId, Guid explanationId)`	
Parameters	`patternId`-[`string`]	Pattern Identifier created using a hash function
	`explanationId`-[`Guid`]	GUID assigned by the CMS repository software
Returns	Boolean	True – row successfully updated the LIE `patterns_explanations` table
Exceptions Thrown	*None*	

Related Information

```
"UPDATE patterns_explanations SET explanation_id='{1}' WHERE
pattern_id='{0}'", patternId, explanationId)
```

DeleteMapping method

LIE.DataAccess Namespace **- Patterns** Class **–** public **DeleteMapping** Method

Purpose	Permits LIE to remove the relationship between a bill statement line item and an explanation.	
Description	1. Use the LIE data source connection string from the constructor 2. Execute the query and test for success	
Synopsis	`public bool DeleteMapping(string patternId, Guid explanationId)`	
Parameters	`patternId`-[`string`]	Pattern Identifier to remove mapping
	`explanationId`-[`Guid`]	GUID of the explanation
Returns	Boolean	True – row deleted from the LIE `patterns_explanations` table
Exceptions Thrown	*None*	

Related Information

```
"DELETE FROM patterns_explanations WHERE pattern_id='{0}' AND
explanation_id='{1}'", patternId, explanationId)
```

RemoteServicesDAC

Summary

Facilitates data access to the `remote_services` LIE database table. This code is still under development for the control console of the LIE application.


```
public RemoteServicesDAC()
public RemoteServicesDataset GetAllServices()
```

Properties

Name	Type	Description

Enumerations or Constants

```
CONNECTION_STRING = "data source=CBTXDB01;initial
catalog=LIE;password=password1;persist security info" +
"=True;user id=lie_dev;workstation id=PPETROV;packet size=4096";
```

Data Members

None

Significant .NET NameSpaces Used

`System.Data`	Build components that efficiently manage data from multiple data sources. Provides the tools to request, update, and reconcile data in multiple-tier systems.
`System.Data.SqlClient`	Collection of classes used by MS SQL Server to access a data source in the managed space.
`LIE.Data`	Generated C# code that defines the objects mapped into relational records.

Configuration Constants

None

RemoteServicesDAC method

LIE.DataAccess Namespace **- RemoteServicesDAC** Class **–** public **RemoteServicesDAC** Method

Purpose	Initialize `cmdSelectAllServices` command
Description	1. Set data source connection string to LIE Production SQL Server 2. Build command to read all information from `remote_services` LIE table
Synopsis	`public RemoteServicesDAC()`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

`"SELECT * FROM remote_services"`

GetAllServices method

LIE.DataAccess Namespace **- RemoteServicesDAC** Class **–** public **GetAllServices** Method

Purpose	Read all information in the LIE production `remote_services` table into an in-memory dataset	
Description	Implement the command to read all information from `remote_services` LIE table	
Synopsis	`public RemoteServicesDataset GetAllServices()`	
Parameters	*None*	

Returns	RemoteServicesDataset	**Each remote_services dataset row holds:**
		`URI (Primary Key)`
		`protocol`
		`host_name`
		`port`
		`service_name`
		`service_type`

Exceptions Thrown	*Remote Services exception*	Console.WriteLine(ex.ToString()

Related Information

```
"SELECT * FROM remote_services"
```

Statement

Summary

Provides access to the LIE and Oracle data sources to retrieve and store billing statements. This class is called from the `WorkFlow` namespace to import an number of bills (1000) within a single Verizon billing jurisdiction.

Methods

`public Statement(string lieConnString, string bvConnString)` – OVERLOADED

`public string GetFromBackend(short fscId, string accountId, ref string statementDate)`

`public bool SaveToLieDb(StatementData statement)`

`public string GetFromLieDb(short fscId, string accountId)`

`public string GetFromLieDb(short fscId, string accountId, DateTime billDate)` – [**Overloaded**]

`public IDataReader GetFromLieDb(short fscId, int numberOfStatements)` – [**Overloaded**]

`public bool IsExistingAccount(string accountId)`

`public LieDbDataSet.accountsRow GetAccount(string accountId, short fscId)`

`public bool InsertAccount(string accountId, string lob, short fscId, string btn, int npa, int nxx, string state, string zip)`

`public LieDbDataSet GetOccurrencesDataset(ArrayList tableNames, short fscId, string lob)`

`public void InsertTable(DataTable table)`

`public void UpdateStatements(string statementId, int occurrenceCount, int lineCount)`

`public void DeleteStatement(string statementId)`

`public int GetNumberOfOccurrences(string statementId)`

Properties

Name	Type	Description

Constants

Line Of Business constant. This component is going to be used by Consumer market only. But since it shares database tables with General Business ('B') application we always need to pass 'C' to get the data we need
`string LOB = "C"`

Data Members

None

Significant .NET NameSpaces Used

`System.Collections`	Interfaces and classes that define collections of objects, such as lists, queues, bit arrays, hash tables and dictionaries.
`System.Configuration`	Access .NET Framework configuration settings and handle errors in configuration (*.config) files.
`System.Data`	Build components that efficiently manage data from multiple data sources. Provides the tools to request, update, and reconcile data in multiple-tier systems.
`System.Data.Common`	Collection of classes used by a .NET data provider to access a data source (database) in the managed space.
`System.Data.SqlClient`	Collection of classes used by MS SQL Server to access a data source in the managed space.
`System.Text`	Classes representing ASCII, Unicode, UTF-7, and UTF-8 character encoding schemes; plus abstract base classes for converting blocks of characters to and from blocks of bytes; and a helper class that manipulates and formats String objects.
`System.Xml`	Provides standards-based support for processing XML
`LIE.Data`	Generated C# code that defines the objects mapped into relational records.
`LIE.Logging`	Log key application events in IIS trace log. Provide output message string that included a developer message, stack information, and Named Values from the developer.
`BillViewObjectR2`	BillView Object class revision #2 coded in Visual Basic.

Configuration Constants

```
"lieDbConnString" value="Network Library=DBMSSOCN;Data
Source=cbtxdb01,1433;Initial Catalog=LIE;Integrated Security=false;User
ID=lie_dev;Password=password1;" />

Name (Type): Description
```

Statement method

LIE.DataAccess Namespace **- Statement** Class **–** public **Statement** Method

Purpose	Get data source connection strings for the LIE Production and BillView Production data sources from the `Web.config` file.

Description	Obtain values for the configuration parameters:
	`BillViewConnString`
	`lieDbConnString`

Synopsis	`public Statement`
Parameters	*None*
Returns	*None*
Exceptions Thrown	*None*

Related Information

```
"lieDbConnString" value="Network Library=DBMSSOCN;Data
Source=cbtxdb01,1433;Initial Catalog=LIE;Integrated
Security=false;User ID=lie_dev;Password=password1;"

"BillViewConnString" value="Server=billtest;Integrated
Security=false;Persist Security Info=false;Pooling=false;Max Pool
Size=100;Min Pool Size=0;User ID=ebillview;password=ebillview_test;"
```

Statement (overloaded) method

LIE.DataAccess Namespace **- Statement** Class **–** public **Statement** Method

Purpose Assign LIE pattern research and BillView database connection strings

Description

Synopsis

```
public Statement(string lieConnString, string bvConnString)
```

Parameters

`lieConnString`-[String]	Production LIE data source connection string
`bvConnString`-[String]	Production BillView data source connection string

Returns *None*

Exceptions Thrown *None*

Related Information

GetFromBackend method

LIE.DataAccess Namespace **- Statement** Class **–** public **GetFromBackend** Method

Purpose	Retrieves billing statement from backend using BillViewObjectR2 through interop.	
Description	1. Create instance of `BillViewObject` if it is not created yet 2. Do not perform any LIE XML additions 3. Try to retrieve statement and throw an exception if unsuccessful 4. The **Load** method always preserves significant white space. The **PreserveWhitespace** property determines whether or not white space is preserved. The default is false, whites space is not preserved. 5. Select the first **XmlNode** that matches the "`/XML-BILL/XML-S/STATEMENT`" expression. 6. Extract the information found in the `Date` attribute for this node	
Synopsis	`public string GetFromBackend(short fscId, string accountId, ref string statementDate)`	
Parameters	`fscId`-[`short`]	Jurisdiction of bill
	`accountId`-[`string`]	Customer Account Number
	`statementDate`-[`string`]	Date of the statement
Returns	string	Statement as an XML string
		Returns actual statement date found in the XML
Exceptions Thrown	*ApplicationException*	"BillView object was not able to retrieve statement:\n" + BillViewObject error Message text)

Related Information

```
Constant - billViewObject.SupportLIEParsing = false

Constant - string groupId = "7005"

GetBillData(accountId, groupId, fscId, statementDate, ref
objStatement, ref objSmry, ref listDates,
BVO_APP_GRP.BV_USERDEF_GRP, addInfo, billType, false)

statementDate = xmlDoc.SelectSingleNode("/XML-BILL/XML-
S/STATEMENT").Attributes["DATE"].Value
```

SaveToLieDb method

LIE.DataAccess Namespace **- Statement** Class **–** public **SaveToLieDb** Method

Purpose	Saves XML statement in the `statements` table of the LIE database	

Description	1. Create an INSERT query
	2. Add the following parameters to INSERT statement
	`statement_id varchar(64)`
	`account_id varchar(20)`
	`statementDate varchar(10)`
	`xmlStatement text`
	`occurence_count integer`
	`line_count integer`
	3. Execute the INSERT and test for success	

Synopsis	`public bool SaveToLieDb(StatementData statement)`	
Parameters	`statement`-[StatementData]	Statement to store into the LIE Research database
Returns	Boolean	True – XML statement successfully added to the `statements` LIE table
Exceptions Thrown	*None*	Should log some message with the statement id

Related Information

```
"INSERT INTO statements VALUES(@statementId, @accountId,
@statementDate, @xmlStatement, @occurrenceCount, @lineCount)"
```

GetFromLieDb method

LIE.DataAccess Namespace **- Statement** Class **–** public **`GetFromLieDb`** Method

Purpose	Retrieves most recent billing statement from LIE database for this account number and jurisdiction.	
Description	Calls `GetFromLieDb` overloaded method with most recent statement date.	
Synopsis	`public string GetFromLieDb(short fscId, string accountId)`	
Parameters	`fscId`-[short]	Jurisdiction number for bill
	`accountId`-[string]	Customer account number
Returns	string	Billing statement in XML-BILL format
Exceptions Thrown	*None*	

Related Information

GetFromLieDb (overloaded) method

LIE.DataAccess Namespace **- Statement** Class **–** public **GetFromLieDb** Method

Purpose	Retrieves billing statement from LIE database for this account number, statement date, and jurisdiction.	
Description	1. Build SELECT statement based on `billDate` input 2. Execute the query 3. Test results and log either a success or failure	
Synopsis	`public string GetFromLieDb(short fscId, string accountId, DateTime billDate)`	
Parameters	`fscId`-[short]	Jurisdiction number for bill
	`accountId`-[string]	Customer account number
	`billDate`-[DateTime]	Statement date
Returns	string	Billing statement in XML-BILL format
Exceptions Thrown	*None*	LogManager writes "Statement query returned null" if the query does not return any information

Related Information

Most recent statement - `"SELECT statement FROM statements WHERE account_id='{0}'", accountId);`

Statement Date Specified - `"SELECT statement FROM statements WHERE account_id='{0}' AND statement_date='{1}'", accountId, billDate.ToString("yyyy-MM-dd")`

GetFromLieDb (multistatement) method

LIE.DataAccess Namespace **- Statement** Class **–** public **GetFromLieDb** Method

Purpose	Retrieves the desired number of billing statements from LIE database for this jurisdiction.	
Description	1. Build query 2. Use `IDataReader` to access a forward-only stream of a result set obtained by executing a command with the LIE data source 3. Throw an exception if an error happens	
Synopsis	`public IDataReader GetFromLieDb(short fscId, int numberOfStatements)`	
Parameters	`fscId`-[short]	Jurisdiction number for these statements
	numberOfStatements-[int]	Number of statements to load to the LIE database
Returns	IDataReader	Returns a datastream of XML statements read from the LIE research `statements` database
Exceptions Thrown	*None*	Throws a nonspecific exception

Related Information

SELECT TOP {0} * FROM statements WHERE account_id IN (SELECT account_id FROM accounts WHERE fsc_id={1})", numberOfStatements, fscId)

IsExistingAccount method

LIE.DataAccess Namespace **- Statement** Class **–** public **IsExistingAccount** Method

Purpose	Checks if the customer account exists in the LIE research pattern identification database	
Description	1. Build SELECT query 2. Execute the query 3. Convert row count to an integer and test for non-zero value	
Synopsis	`public bool IsExistingAccount(string accountId)`	
Parameters	accountId-string]	Customer account number
Returns	boolean	True – account number exists in the LIE research pattern identification database
Exceptions Thrown	*None*	

Related Information

```
"SELECT COUNT(*) FROM accounts WHERE account_id='{0}'", accountId)
```

GetAccount method

LIE.DataAccess Namespace **- Statement** Class **–** public **`GetAccount`** Method

Purpose	Get customer account information from the BillView Oracle database	
Description	1. Build Oracle SELECT statement 2. Execute the Oracle reader	
Synopsis	`public LieDbDataSet.accountsRow GetAccount(string accountId, short fscId)`	
Parameters	`accountId`-[string]	Customer account number
	`fscId`-[short]	Jurisdiction indicator
Returns	`LieDbDataSet.accountsRow`	Account information values accountId – Customer account fscId – Jurisdiction btn – BillingTelephone Number npa – 3-char area code nxx – 3-char telephone prefix state – 2 character state designation zip = zipcode
Exceptions Thrown	*Oracle Reader*	`"Exception occured while trying to retrive account information from the BillView db for account " + accountId + " and fscId " + fscId`

Related Information

"SELECT cust_btn, state, zip FROM reg_btn WHERE account_id ='{0}' AND fsc_id={1}", accountId, fscId

InsertAccount method

LIE.DataAccess Namespace **- Statement** Class **–** public **`InsertAccount`** Method

Purpose	Inserts record into the LIE research pattern identification `accounts` table	
Description	1. Build SQL server query 2. Execute the query and test rows affected 3. Return true value if one or more rows affected	
Synopsis	`public bool InsertAccount(string accountId, string lob, short fscId, string btn, int npa, int nxx, string state, string zip)`	
Parameters	`accountId`-[`string`]	Customer Account Number
	`lob`-[`string`]	Level of Business [**C**-Consumer | **GB**-General Business]
	`fscId`-[`short`]	Jurisdiction indicator
	`btn`-[`string`]	Billing Telephone associated with customer account
	`npa`-[`int`]	Area code
	`nxx`-[`int`]	Calling prefix within this area code
	`state`-[`string`]	State where service is rendered
	`zip`-[`string`]	Zipcode where service is rendered
Returns	boolean	Record inserted into the `accounts` LIE research table
Exceptions Thrown	*None*	

Related Information

```
"INSERT INTO accounts
VALUES('{0}','{1}',{2},'{3}',{4},{5},'{6}','{7}')", accountId, LOB,
fscId, btn, npa, nxx, state, zip)
```

GetOccurrencesDataset method [Obsolete]

LIE.DataAccess Namespace **- Statement** Class **–** public **GetOccurrencesDataset** Method

Purpose	For a given Level of Business and Jurisdiction (Reports) copy rows from from the LIE research pattern identification database tables: `occurrences, line_items, content, context`	
Description	1. Create connection and SqlDataAdapter to LIE database 2. For each table name in the array 3. Build SELECT statement for the table 4. Fill the in-memory dataset	
Synopsis	`public LieDbDataSet GetOccurrencesDataset(ArrayList tableNames, short fscId, string lob)`	
Parameters	`tableNames`-[`ArrayList`]	Array of LIE database table names occurences line_items content context
	`fscId`-[`short`]	Jurisdiction indicator
	`lob`-[`string`]	Level of Business [**C**-Consumer | **GB**-General Business]
Returns	LieDbDataSet	In-memory representation
Exceptions Thrown	*None*	

Related Information

`"SELECT * from {0} WHERE fsc_id={1} AND lob={2}", tableName, fscId, lob)`

Property: `dsLieDb.EnforceConstraints` = false during method actions

InsertTable method

LIE.DataAccess Namespace **- Statement** Class **–** public **`InsertTable`** Method

Purpose	Saves entire table content to the LIE database	
Description	1. Create connection and SELECT statement to bring all previous table rows into memory 2. Update the table completely 3. For any exceptions in a table row – log the exception string created from the table name, table column name and the actual column value	
Synopsis	`public void InsertTable(DataTable table)`	
Parameters	`table`-[`DataTable`]	LIE database table to be updated
Returns	*None*	
Exceptions Thrown	*Update Table* *ApplicationException*	`Exception occured while trying to save table: " + table.TableName + Table Column Name + Table Column Value` `Logging`

Related Information

```
"select * from " + table.TableName, cn
```

UpdateStatements method

LIE.DataAccess Namespace **- Statement** Class **–** public **`UpdateStatements`** Method

Purpose	Updates `occurrence_count` and `line_count` fields in the LIE research `statements` table for a given `statement_id`	
Description	1.	
Synopsis	`public void UpdateStatements(string statementId, int occurrenceCount, int lineCount)`	
Parameters	`statementId`-[`string`]	Statement Identifier (account ID, billdate, and jurisdiction concatenation)
	`occurrenceCount`-[`int`]	Number of occurrences of this pattern in am statement
	`lineCount`-[`int`]	Number of identified patterns in this statement
Returns	*None*	
Exceptions Thrown	*None*	

Related Information

```
"UPDATE statements SET occurrence_count={1}, line_count={2} WHERE
statement_id='{0}'", statementId, occurrenceCount, lineCount
```

DeleteStatement method

LIE.DataAccess Namespace **- Statement** Class **–** public **`DeleteStatement`** Method

Purpose	Delete a statement record from the LIE research `statements` and `occurrences` table	
Description	1. Build the queries 2. Execute both DELETE queries shown below 3. Commit the DELETE transactions 4. On exception, log the statement identifier that failed	
Synopsis	`public void DeleteStatement(string statementId)`	
Parameters	`statementId`-[`string`]	Statement Identifier (account ID, billdate, and jurisdiction concatenation)
Returns	*None*	
Exceptions Thrown	*Delete*	Exception occured while trying to purge a statement " + statementId + ex3.ToString()

Related Information

```
"DELETE FROM occurrences where statement_id='" + statementId + "'"
"DELETE FROM statements WHERE statement_id='" + statementId + "'"
```

GetNumberOfOccurrences method

LIE.DataAccess Namespace **- Statement** Class **–** public **GetNumberOfOccurrences** Method

Purpose	Returns number of occurrences (patterns) in the LIE research `statements` table for this statement ID (if it was decomposed)	
Description	1. Create connection and SELECT query 2. Execute query 3. If successful – return number of occurences 4. Throw exception if failure	
Synopsis	`public int GetNumberOfOccurrences(string statementId)`	
Parameters	`statementId`-[`string`]	Statement Identifier (account ID, billdate, and jurisdiction concatenation)
Returns	int	Number of pattern occurrences in a single statement
Exceptions Thrown	*None*	`"Error occurred while trying to get number of occurrences from the statement table"`

Related Information

"SELECT occurrence_count FROM statements WHERE statement_id='" + statementId + "'"