MIMOSA OSA-EAI

TECH-CDE CLIENT-SERVER XML SCHEMA

PRIMER

VERSION 3.2.3

LICENSE AGREEMENT

NOTICE TO USER: CAREFULLY READ THE FOLLOWING LEGAL AGREEMENT (THE“AGREEMENT”). YOUR USE OR DOWNLOADING OF ANY SOFTWARE, TOOLS, SPECIFICATIONS OR DOCUMENTATION (COLLECTIVELY, THE “MATERIALS”) MADE AVAILABLE TO YOU BY THE MACHINERY INFORMATION MANAGEMENT OPEN SYSTEMS ALLIANCE (“MIMOSA”) CONSTITUTES YOUR ACCEPTANCE OF THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN OR DESTROY THE MATERIALS.

1. License and Restrictions. MIMOSA grants to you a non-exclusive, royalty-free, perpetual license under MIMOSA’s rights in the Materials to use and reproduce the Materials for your internal use. You may not sublicense the rights granted herein without the prior written consent of MIMOSA. You agree not to engage in or encourage the reverse engineering or reverse compilation of any object code included in the Materials.

2. Ownership of Intellectual Property. You acknowledge that title and full ownership rights to the Materials, including all proprietary rights therein, will remain the exclusive property of MIMOSA and its licensors, and that you will not acquire any rights to the Materials except as expressly set forth above. You agree not to remove any product identification, copyright or other proprietary notice from the Materials.

3. Warranties. You acknowledge that the Materials have been the result of a collective effort of several MIMOSA member organizations who have donated their time and resources and, as such, MIMOSA and its licensors make no warranty or representation, express or implied, with respect to the Materials. You agree to indemnify and hold MIMOSA harmless from and against all liabilities, losses, damages, costs and expenses, including attorneys’ fees, which MIMOSA may incur or otherwise suffer as a result of your use of the Materials.

YOU ACKNOWLEDGE THAT THE MATERIALS LICENSED HEREUNDER IS PROVIDED “AS IS,” WITHOUT WARRANTY OF ANY KIND. MIMOSA SPECIFICALLY DISCLAIMS, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY RIGHTS.

4. Limitation of Liability. IN NO EVENT SHALL MIMOSA OR ITS LICENSORS BE LIABLE TO YOU OR ANY OTHER PERSON OR ENTITY FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL, INDIRECT, EXEMPLARY OR PUNITIVE DAMAGES, HOWEVER CAUSED, WHETHER FOR BREACH OF CONTRACT, TORT, NEGLIGENCE, STRICT PRODUCT LIABILITY OR OTHERWISE (INCLUDING, WITHOUT LIMITATION, DAMAGES BASED ON LOSS OF PROFITS, DATA OR BUSINESS OPPORTUNITY), AND WHETHER OR NOT SUCH PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN.

5. Term and Termination. The term of this license is for the duration of any copyright in the Materials. You may terminate this Agreement at any time by written notice to MIMOSA. This Agreement shall terminate immediately without notice if you fail to comply with any provision of this Agreement. Upon termination you shall immediately cease using the Materials and dispose of the Materials.

6. Government Contracts. If you are an agency or instrumentality of the United States Government, the Materials are “commercial computer software” and “commercial computer software documentation”, and pursuant to FAR 12.212 or DFARS 227.7202, and their successors, as applicable, use, reproduction and disclosure of the Materials is governed by the terms of this Agreement.

7. Export Restrictions. You acknowledge that the laws and regulations of the United States restrict the export and re-export of certain commodities and technical data of United States origin, which may include the Materials, in any medium. You agree that you will not knowingly, without prior authorization if required, export or re-export the Materials in any medium without the appropriate United States and foreign government licenses.

8. General Provisions. This Agreement shall be governed, construed and enforced in all respects by the laws of the State of California, without respect to its conflict of laws provisions. The United Nations Convention on Contracts for the International

Sale of Goods is specifically excluded from application to this Agreement. In the event any action is brought for any breach or default of any of the terms of this Agreement, or otherwise in connection with this Agreement, the prevailing party shall be entitled to recover from the other party all costs and expenses (including reasonable attorneys' fees) incurred therefrom. The relationship of the parties established by this Agreement is that of licensor and licensee, and nothing contained herein shall be construed to constitute either party as the agent of the other party as partners, joint ventures, co-owners or otherwise as participants in a joint or common undertaking. Neither party shall be liable hereunder by reason of any failure or delay in the performance of its obligations hereunder (except for the payment of money) on account of strikes, riots, insurrection, fires, flood, storm, explosions, war, governmental action, labor conditions, earthquakes, material shortage, or any other cause which is beyond the reasonable control of such party. You shall not assign or otherwise transfer any of your rights or obligations under this Agreement without MIMOSA's prior written consent. All terms of this agreement shall be binding upon, inure to the benefit of, and be enforceable by the parties hereto and their respective heirs, legal representatives, successors, and assigns. No failure or delay on the part of either party to exercise, in whole or in part, any right or privilege hereunder shall operate as a waiver thereof or of any right to exercise or enforce such right or any other right or privilege hereunder. This Agreement constitutes the entire agreement between the parties regarding its subject matter and supersedes and its terms govern, all prior proposals, agreements or other communications between the parties regarding such subject matter. Should any part of this Agreement be held to be invalid by a court of competent jurisdiction, the remainder of this Agreement shall be considered as the whole and be binding on the parties. Should you have any questions concerning this Agreement, or if you desire to contact MIMOSA for any reason, please write: MIMOSA, 31882 Paseo Alto Plano, San Juan Capistrano, California, 92675-3406 (Tel: (+1-949-625-8616, Fax: +1-949-625-8616)

TECH-CDE CLIENT-SERVER XML SCHEMA PRIMER VERSION 3.2.2

ABOUT TECH-CDE

The Tech-CDE (Compound Document Exchange) Client & Server XML Schema was developed as a means of transferring large sets of CRIS data in XML format using one client-server query schema and one client-server write schema. Because the client has access to a broad query range, normally a server system will natively access CRIS content.

The Tech-CDE Specification consists of two main schemas, a query and a write schema. There is also a supporting schema, V3-2 TechCDE_CrisVocabAddition.xsd, which contains elements used by both schemas. This supporting schema defines all of the entities defined by the MIMOSA Common Relational Information Specification (CRIS). The *Tech*-CDE schema is independent of any "connect" or "disconnect" methods for a client and server system.

COMMON ELEMENTS IN TECH-CDE SCHEMAS

The Tech-CDE write and query schemas are described in more detail later in the document. The following are elements common to both schemas.

HEADER

Both the Tech-CDE query and write schema include a standard header element that must be included with every request message. The header consists of a session identification which is required and 4 optional parameters which are now explained in greater detail.

It should be noted that the 4 optional parameters are set to 0 by default; and only need to be entered if the parameter is being changed to a value of 1. The 4 optional parameters are only applicable to Tech-CDE query requests, and the server is not required to implement the functionality.

SESSION_ID

The parameter **session_id is** an identification tag that is sent with the request. The ID is sent back with the acknowledgement, allowing the client application to correlate responses with acknowledgements.

Example: **<header session_id="34"/>**

Although in this example, the session_id is a number and most likely the number of the query that was sent to the server; this session_id could be of any value that is deemed useful. For instance, the client could send the server a session_id which consists of a timestamp, thus keeping track of each session in the event of an error.

Example: **<header session_id="04-29-2008 13:23:04"/>**

INCLUDE_NON_ACTIVE_ROWS

The parameter include_non_active_rows is a Boolean flag that is used to indicate whether or not the server should transmit rows which have a row status type of not active. Not active rows are inactive and are also referred to as “soft-deleted”. Any row without an rstat_type_code of ‘1’ is considered non active.

Example: **<header include_non_active_rows = ‘1’/>**

INCLUDE_CRIS_REF_DATA_ROWS

The parameter Include_CRIS_ref_data_rows is a Boolean flag used to indicate whether the server should transmit MIMOSA CRIS Reference Data Library rows to the client. These rows provide supporting type data about a row of EAI information. (i.e. for a segment row, a ref data row would be the segment_type)

Example: **<header include_non_active_rows = ‘1’/>**

INCLUDE_ROW_INFO_COLUMNS

The parameter include_row_info_columns is a Boolean flag that is used to indicate whether the server should transmit the following columns in return rows:

- gmt_last_updated
- last_upd_db_site
- last_upd_db_id
- rstat_type_code

These attributes/columns exist for every CRIS entity and are optional.

Example: **<header include_row_info_columns= ‘1’/>**

INCLUDE_LC_INFO_COLUMNS

The parameter include_lc_info_columns is a Boolean flag that is used to indicate whether the server should transmit local columns using the optional column “lcinfo”. Lcinfo columns are defined by the client and are not part of the CRIS specification.

Example: **<header include_lc_info_columns= ‘1’/>**

FILTER TYPES

The “filterTYPE” element is used to filter data based on column and value. This is used for write (update, delete) and query. The following filter types are supported.

- equal_value

- notequal
- min_value
- max_value
- like_value

Example: **<filterTYPE column_name="segment_site" equal_value="000003F900000001" />**

In this example, the filterTYPE column name is set to segment_site and the value must be equal to 000003F9000000001.

Additionally the column name must be specified in the filter type. One option would be to just have the column name with no filter criteria; however, this would result in all of those columns being deleted or edited.   An example of this would be editing the data value of a table to zero for all rows, thereby resetting the value back to zero.

TECH-CDE WRITE SCHEMA

INTRODUCTION

When using Tech-CDE there are two main schemas, as previously mentioned, write and query.

The Tech-CDE Write Schema is used for inserting, altering /updating, soft deleting or hard deleting information in the OSA-EAI Formation. While this schema is most often used in conjunction with databases such as SQL, MySQL and Oracle it could be used for additional data sources.   When the write schema is used, communications that take place with the data-source have the availability to be rolled back if errors occur.

When using the write schema, the server will also be given an acknowledgement which contains either a message verifying the success of the write; or various messages that can be used for debugging purposes.

The Tech-CDE Write Schema which always contains the element **<mim_write>** is followed by **<mim_write_req>** when being submitted to the server; and contains **<mim_write_ack>** when an acknowledgement is sent back from the server.   Both follow a very general format and allow for great flexibility in write statements.

In a request sent to the server (**<mim_write_req>**) there are 3 parameters that must be included, : header, param and mimosa_rows. Header was already described.

PARAM

The element Param consists of two required parameters: action and transaction class.

Example: **<param action="Insert" trans_class = "Atomic" />**

In this brief example, the action of the write query is to insert data into the database and the transaction class is Atomic. The parameter options are discussed in greater detail below.

ACTION

The parameter action, determines what kind of action will occur to the data source. The available actions available are: insert, update, hard delete and soft delete. These actions are inspected below in greater detail.

- Insert: The insert parameter is used for SQL Insert Statements. When using the parameter insert, all required fields must be passed through to assure the SQL statement does not error out.
    - All primary and foreign keys must be entered
    - All Not NULL fields must be included
    - Duplicate primary keys are not valid
- Update: The update parameter is used for SQL alterations. This is used to change data column values in an already existing row.
    - All fields that wish to be updated must also be entered
- Hard Delete: The hard_delete parameter is used to remove SQL data.
    - Dependent rows (with foreign keys to the parent) must be deleted prior to delete the parent row. The rows higher in the structure must be specified first. The statements in a hard delete request run in the reverse order they are specified; ‘bottom up’.
- Soft Delete: The soft_delete parameter is used for soft deleting informaiton. The soft_delete parameter changes the field rstat_type_code to 3.
    - Does not require dependent FK entries to be deleted.

TRANSACTION CLASS

The parameter trans_class determines what the server should do when there is an error. There are two options that are available in a request sent by the client, those being atomic and partial. Those options are explained in greater detail.

- Atomic
    - The server shall rollback changes. If an error occurs part way through the transactions, the SQL server will receive the rollback command and the prior changes will not be committed.
- Partial
    - The server shall not rollback any changes. If the client had sent several queries to be executed, and one error, the server will continue processing the data and commit any changes that do not error. There will be no rollback; the client however will be given error information to address the problem at a later time.

MIMOSA_ROWS

The last request element is mimosa_rows which can contain any and all of the OSA-EAI entity rows with any or all information of the entities attributes. What information is required with a row is not specified in the schemas because of the complexity of the different functionalities.

EXAMPLE INSERT REQUEST

```
<mim_write xmlns=”http://www.mimosa.org/TechCDEV3-2”>
   <mim_write_req>
      <header session_id = “34” />
      <param action=”Insert” trans_class = “Atomic” />
      <mimosa_rows>
            <site site_code="0033F0B100000005" rstat_type_code="1" enterprise_id="3403953" site_id="5"  user_tag_ident="LAV_50"
             template_yn="N" st_db_id="2" st_db_site="0000000000000000" st_type_code="1" name="Light Armored Vehicle 50"/>
           <segment name="Battery1" segment_group_yn="N" segment_id="0" segment_site="0033f0b100000005" sg_db_id="0"
           sg_db_site="0033f0b100000000" sg_type_code="4" user_tag_ident="Battery1" rstat_type_code="1"/>
     </mimosa_rows>
</mim_write_req>
<mim_write>
```

SQL STATEMENTS GENERATED BY THE REQUEST

```
INSERT INTO site (site_code , rstat_type_code , enterprise_id , site_id , user_tag_ident , template_yn ,
st_db_id , st_db_site , st_type_code , name ) VALUES( ‘0033F0B1000000005’ , ‘1’ , ‘3403953’ , ‘5’ ,
‘LAV_50’  , ‘N’ , ‘2’ , ‘0000000000000000’ , ‘1’  , ‘Light Armored Vehicle 50’);

INSERT INTO segment (name , segment_group_yn , segment_id , segment_site , sg_db_id , sg_db_site ,
sg_type_code , user_tag_ident , rstat_type_code) VALUES (‘Battery1’ , ‘N’ , ‘0’ , ‘0033F0B1000000005’ , ‘0’
, ‘0033f0b100000000’ , ‘4’ , Battery1’ , ‘1’);
```

In this example, two insert requests are being sent to the server. The session id of 34 is being used as a unique identifier which can be later used for debugging and logging information.

The insert has a transaction type of “Atomic”. This transaction type deems it necessary for both rows to be successfully entered into the database or a roll-back will occur and none of the data will be committed. In this example if the segment row was not entered properly, the database would roll back and the site would be removed. In the event that the transition class would have been set to Partial, then any data that was entered would not have been rolled back.

EXAMPLE INSERT RETURN

The response that should be returned from the Server, if all information is returned correctly is as follows.

```
<mim_write xmlns="http://www.mimosa.org/TechCDEV3-2">
            <mim_write_ack>
                        <header session_id="34"/>
```

```
                    <status success="1" message_code="" message_text=""/>
          </mim_write_ack>
</mim_write>
```

In this response we can see that the server sends an acknowledgement. In this acknowledgement, the session_id that is sent is the same session_id that was entered in the request. This is helpful as previously stated in debugging and logging information as one would be able to determine which queries had potential issues.

Additionally the status success parameter was a 1 which indicates that the insert was successful. In the event that the insert was unsuccessful, the server may have attached error information in the message code field. The error message code is composed of 3 parts, which is used for debugging information and generally takes the format of "0000000000000000-nnnn". For additional information in regards to what the error message consists of, consult the Cris_Vocab_Addition.xsd file.

EXAMPLE UPDATE REQUEST

```
<mim_write xmlns=”http://www.mimosa.org/TechCDEV3-2”>
   <mim_write_req>
      <header session_id = “35” />
      <param action=”Update” trans_class = “Atomic” />
      <mimosa_rows>
            <segment_chr_data data_value = “000000x0”>
               <filterTYPE column_name="segment_site"    equal_value="000003F900000001" />
              <filterTYPE column_name="segment_id"       equal_value="21" />
              <filterTYPE column_name=”sc_db_site”        equal_value=”0000000000000000”/>
              <filterTYPE column_name=”sc_db_id”          equal_value =”0” />
              <filterTYPE column_name=”sc_type_code”     equal_value = “0” />
              <filterTYPE column_name=”eu_db_site”        equal_value = “0000000000000000”/>
              <filterTYPE column_name=”eu_db_id”          equal_value = “0” />
              <filterTYPE column_name=”eu_type_code”    equal_value= “0”/>
         </segment_chr_data>
     </mimosa_rows>
</mim_write_req>
</mim_write>
```

SQL STATEMENT GENERATED BY THE REQUEST

```
UPDATE  segment_chr_data  SET data_value = ‘000000x0’ WHERE segment_site = ‘000003F900000001’
AND segment_id = ‘21’ AND sc_db_site = ‘0000000000000000’ AND sc_db_id = ‘0’ AND sc_type_code = ‘0’
AND eu_db_site = ‘0000000000000000’ AND eu_db_id = ‘0’ and eu_type_code = ‘0’;
```

In the above example, the client is sending a request to update a row in the table segment_chr_data. The first step taken is that the client appends the new information to the segment_chr_data tag, which is evident by the request containing ‘data_value = “000000x0” ’

Then, the filterTYPE element is added to specify which rows should be updated. In the example, each filterTYPE contains the column name and the value it should be equal to.

EXAMPLE DELETE REQUEST

```
<mim_write xmlns=”http://www.mimosa.org/TechCDEV3-2”>
   <mim_write_req>
      <header session_id = “36” />
      <param action=”Hard_Delete” trans_class=”Atomic” />
      <mimosa_rows>
            <segment_chr_data>
               <filterTYPE column_name="segment_site"   equal_value="000003F900000001" />
              <filterTYPE column_name="segment_id"      equal_value="21" />
              <filterTYPE column_name=”sc_db_site”      equal_value=”0000000000000000”/>
              <filterTYPE column_name=”sc_db_id”        equal_value =”0” />
              <filterTYPE column_name=”sc_type_code”     equal_value = “0” />
              <filterTYPE column_name=”eu_db_site”      equal_value = “0000000000000000”/>
              <filterTYPE column_name=”eu_db_id”        equal_value = “0” />
              <filterTYPE column_name=”eu_type_code”   equal_value= “0”/>
          </segment_chr_data>
     </mimosa_rows>
</mim_write_req>
</mim_write>
```

SQL STATEMENT GENERATED BY THE REQUEST

```
DELETE FROM segment_chr_data WHERE segment_site = ‘000003F900000001’ AND
segment_id = ‘21’ AND sc_db_site = ‘0000000000000000’ AND sc_db_id = ‘0’ AND sc_type_code = ‘0’
AND eu_db_site = ‘0000000000000000’ AND eu_db_id = ‘0’ and eu_type_code = ‘0’;
```

In the above example, the server is sent a request to delete a row from the segment_chr_data table. To assure only one row is deleted, the primary key fields are passed in using the filterTYPE element . The above example only deletes data from one table and one row, but a request can specify many tables and many rows.

MESSAGE TEXTS SENT BY SERVER

Furthermore, the message text would also consist of debugging information. The following message texts are currently used by MIMOSA. However, the server is free to add additional messages as needed.

- "Invalid Connect String"
- "Data Source Not Available"
- "Data Limit Exceeded"
- "No Privilege For Operation"
- "Language Not Supported"
- "Function Not Implemented"
- "Cannot Create Entry - Primary Key Already Exists"
- "Cannot Create Entry - Not All Required Data Specified"
- "Cannot Create Entry - Invalid Data Specified"
- "Cannot Create Entry - Exceeded Attachment Limit"
- "Cannot Create Entry - Attachment Data Type Not Supported"
- "Session ID Not Valid - Connect Required"

TECH-CDE QUERY SCHEMA

The Tech-CDE Query Schema provides a method of querying a CRIS data source regardless of technology type and leverages the relationships between CRIS entities.

The main attribute of the Tech-CDE Query Schema is <mim_query> which can contain either a **<mim_query_req>** (client request) or **<mim_query_ack>** (server acknowledgement). A request contains a **<header>**, one-to-many **<base>** or **<recursive base>** elements, and a **<compression>** element.

BASE

The main functionality of the query comes from base and recursive base. These elements allow the client to specify the data in which to return. The base element is used for basic querying while recursive base allows querying of parent/child relationships.

BASE PARAMETERS AND RECURSIVE BASE PARAMETERS

The base parameters and recursive base parameters fields allow for different parameters for the query to be specified. The parameters are explained below in greater detail.

- return_fk_tables: (The server does not have to support this capability.)
  - The field return_fk_tables is a Boolean flag that is used to specify whether or not the sever should return the direct foreign key rows of the main query rows
    - The value 0 sets the flag to false
    - The value 1 sets the flag to true
- count_only:
  - The field count_only is a Boolean flag that is used to specify whether or not the sever should only return the count of the rows. This would return the number of rows but none of their respective fields or content.
- last_n_data :
  - The parameter last_n_data can be left null; however, if the value was set to something greater than 0 then only that number of specified rows shall be returned.
    - This is often used in conjunction with the parameter order_by when looking for the last several events in a given time span
- return:
  - The return field is a Boolean flag that is used to determine if the main rows should be returned
    - The value 0 sets the flag to false
    - The value 1 sets the flag to true
- order_by:
  - The parameter order_by can be left null; however, if not left null, the client can put the name of any column of the returned table they wish the results to be ordered by.
    - Examples of this would be gmt_assessment , segment_id , etc
    - This is often used in conjunction with last_n_data
- return_class : This is only for recursive_base queries

  - If the return_class is set to “child , children rows of the matching row should be returned
  - If the return_class is set to “parent”, then parent rows of the matching row should be returned
  - If the return_class is set to “top_parent” , then only rows that are not children of other rows should be returned

COMPRESSION

The compression element allows the client to tell the server that the results of a query should be compressed and stored locally for later retrieval. A Tech-CDE server is not required to implement this functionality.

- compress: Boolean flag indication whether server should compress results
  - The value 0 sets the flag to false
  - The value 1 sets the flag to true
- compress_type: Attribute to specify the type of compression.
  - Zip: zip format
  - Rar: rar format
  - Tar: tar format
- file_name: Location to save compressed file.
- log_file_name: Name of the log file to save outcome of compression
- log_file_location: Location of the log file.

BASE_ROWS AND RECURSIVE_BASE_ROWS

The main part of the query is row; this is where the main query is specified and contains all of the rows that are applicable for each query type.

- In **<base_rows>** all rows are present;
- In **<recursive_base_rows>** only tables that consist of a parent/child relationship are available

RECURSIVE BASE INFO

This element is only used with **<recursive_base>** to specify which table contains the parent / child relationship information. For example, segments can be linked by either segment_child or sg_network_connect; this element allows the request to specify which table the relationship should be found in.

For further information in respect to relationships, please consult the file “V3-2TechCDE_ServerInfo.xml”

RETURN ROWS

The most important aspect of the Tech-CDE Query is the return_rows field. Return_rows allows the client to specify additional rows to be returned based of information based on the main_query they specified.

For example, if the main query was for a site, the server could return child information such as the segments that belonged to that site. The schema allows for any row to be specified, but for the query to work properly the rows must have a direct foreign key relationship with the main query.

It should also be noted that return rows could ask for multiple return rows. For example, if the main query was for sites, then the return could consist of site_database and segment. Furthermore, return rows can have their own returns, allowing for that segment to have a return of segment event data. This allows for complex data querying, while only requiring minimum data to be sent from the client.

EXAMPLE QUERY 1

```
<mim_query xmlns="http://www.mimosa.org/TechCDEV3-2">
<mim_query_req>
 <base>
    <base_params return_fk_tables="0" return="0" last_n_data="1" order_by="gmt_installed" />
    <base_rows>
      <asset_on_segment>
        <filterTYPE column_name="segment_site" equal_value="000003F900000001" />
        <filterTYPE column_name="segment_id" equal_value="21" />
      </asset_on_segment>
    </base_rows>
    <return_rows>
      <asset return=”1”/>
    </return_rows>
  </base>
 </mim_query_req>
</mim_query>
```

EXAMPLE OF SQL STATEMENTS GENERATED BY QUERY 1

Some Tech-CDE Servers will be accessing a native MIMOSA OSA-EAI CRIS relational database. When receiving a client Tech-CDE query, the normal method to access the database is the generate SQL statements. This section documents some example SQL statements which would match the example queries shown above. NOTE: Server implementations may generate different syntax from the example SQL statements shown in this document, as long as servers generate the proper XML return to a query.

The SQL statements that are generated from the request above are twofold. The first query is as follows:

```
SELECT * FROM asset_on_segment WHERE segment_site = ‘000003F9000000001’ AND segment_id = ‘21’
```

This query is the result of the criteria set forth in base_rows, to select all data from the table **asset_on_segment** where the criteria holds true. When this data is gathered by the server, the resulting rows are then parsed through and an additional query is preformed on each row. That query is as follows:

```
SELECT * from asset WHERE asset_org_site = ‘results.key’ AND asset_id = ‘results.key’
```

To summarize the queries above: the following is asking to first select (but not to return) the most recently installed row in the **asset_on_segment** table for a specific segment with matching **segment_site** and **segment_id** values of "000003F900000001" and "21". Then for this result, return the foreign key asset row which matches the selected asset_on_segment row.

This allows the client to start with more generic information such as a segment site and segment ID and then get all of the assets that belong to that data. Moreover, not returning all of the information minimizes the acknowledgement reducing the stress on the server.

Along with specifying the **<return_rows>** for the main query, return rows can be specified for each **<return_rows>** element. This allows an unlimited number of **<return_rows>** and depth to be added to a request. All of the **<return_rows>** elements act the same way, they return rows based on the rows returned by their parent.

EXAMPLE QUERY 2

To further demonstrate and explain how to best leverage the Tech-CDE schema, the following example will show how to use multiple return rows and parameters.

```
<mim_query xmlns="http://www.mimosa.org/TechCDEV3-2">
<mim_query_req>

<base>
    <base_params return="0" return_fk_tables="0" last_n_data="10" order_by="gmt_last_updated"/>
     <base_rows>
        <site name ="Example Site"/>
     </base_rows>
     <return_rows>
       <segment return="0">
         <return_rows>
           <meas_location return="0">
             <return_rows>
               <meas_event return="1"/>
             </return_rows>
           </meas_location>
         </return_rows>
       </segment>
     </return_rows>
</base>

 </mim_query_req>
</mim_query>
```

EXAMPLE OF SERVER-INTERNAL SQL STATEMENTS GENERATED BY QUERY 2

```
SELECT * FROM site WHERE name = ‘example site’ LIMIT 10;

SELECT * FROM segment WHERE segment_site = ‘return_row.FK’;

SELECT * FROM meas_location WHERE meas_loc_site = ‘return_row.fK’;

SELECT * FROM meas_event WHERE meas_loc_site = ‘return_row.fk’ AND meas_loc_id = ‘return_row.fk’;
```

In the above example, another basic query is being sent to the server. However, it is important to notice the criteria set in the main tag. There are zero rows being returned from the server to the client; however, the server needs to determine and use the last 10 rows based off of gmt_last_updated for further querying.

In this example the 10 most recent sites added will be stored by the server. From this information the server will retrieve but not return all of the segments that belong to that site, as indicated by the parameter ‘return=”0”’. This information is then further broken down as the measurement locations are being queried but not returned. Finally that information is queried and the measurement events are queried and returned to the client.

This query allows a request sent by the client to start with very broad search criteria, the site, and yet be able to search all the way down to a very specific entity; in this case the measurement locations. This also allows the client to retrieve a broad amount of information in one request; as opposed to having to manually retrieve information from the database with advanced queries that could become complicated and troublesome

EXAMPLE OF QUERY 2 XML RESPONSE

```
<mim_query xmlns="http://www.mimosa.org/TechCDEV3-2">
   <mim_query_ack>
      <header session_id="1" />
      <status success="1" message_text="Success" />
      <row>
             <meas_event dqual_type_code="2" gmt_event="2004-10-15 14:00:00.0" meas_loc_id="1"
             meas_loc_site="0033f0b100000005" ml_db_id="1" ml_db_site="0000000000000000" ml_type_code="15"
             rstat_type_code="1" />
            <meas_event dqual_type_code="2" gmt_event="2004-10-15 18:00:00.0" meas_loc_id="1"
             meas_loc_site="0033f0b100000005" ml_db_id="1" ml_db_site="0000000000000000" ml_type_code="15"
             rstat_type_code="1" />
            <meas_event dqual_type_code="2" gmt_event="2004-10-16 14:00:00.0" meas_loc_id="1"
              meas_loc_site="0033f0b100000005" ml_db_id="1" ml_db_site="0000000000000000" ml_type_code="15"
              rstat_type_code="1" />
     </row>
  </mim_query_ack>
</mim_query>
```

The following query response is the result that would have been seen had the query above been sent to the server. It should be noted that the session_id that was sent to the server is returned in the response, just as previously seen in other queries.

The meas_event elements shown above do not contain all of the fields in the meas_event entity. The server should return all required fields plus any additional fields that have data associated with them.

It should also be noted that neither the site, segment, nor measurement location data are displayed in the return acknowledgement that was sent from the server. This allows the client to have the data desired readily available without needing to drudge through data that may be deemed unnecessary at the current time. This also is more efficient as the data being sent to and from the server is reduced.

KEYCOLUMNS FILTER FOR RETURN ROWS

The use of the query return rows is the means to SQL join associated tables. When the same table has multiple foreign keys to the same table, the join can be come ambiguous. The keyColumns parameter, selects the specific columns to join.

EXAMPLE QUERY 3 WITH KEYCOLUMNS

In this example, the work order table has three agent possibilities: "by" agent, "system" agent, and "for" agent. The keyColumns parameter, selects the specific columns to join. The form is keyColumns = “list of column names”. The columns names are the Foreign Key columns.

```
<mim_query xmlns="http://www.mimosa.org/TechCDEV3-2">
   <mim_query_req>
          <base>
             <base_params return="1" return_fk_tables="0" count_only="0" />
                <base_rows>
                    <work_order return ="1">
                        <filterTYPE column_name="work_order_db_site" equal_value="0000043300000111" />
                    </work_order>
                </base_rows>
                <return_rows>
                    <agent keyColumns= "by_agent_site by_agent_id"  return="1"/>
                </return_rows>
          </base>
   </mim_query_req>
</mim_query>
```

EXAMPLE QUERY 3 RESPONSE

```
<mim_query xmlns="http://www.mimosa.org/TechCDEV3-2">
   <mim_query_ack>
     <header session_id="1" />
     <status success="1" message_text="Success" />
     <row>
          <agent org_agent_site="0000041D00000001" agent_id="1" agent_db_site="0000000000000000" agent_db_id="1"
         agent_type_code="1" user_tag_ident="Test Agent 1" name="Test Agent 1" rstat_type_code="1" />
          <agent org_agent_site="0000041D00000001" agent_id="1" agent_db_site="0000000000000000" agent_db_id="1"
         agent_type_code="1" user_tag_ident="Test Agent 1" name="Test Agent 1" rstat_type_code="1" />
     </row>
  </mim_query_ack>
</mim_query>
```

EXAMPLE QUERY 4 RECURSIVE BASE REQUEST

```
<mim_query xmlns="http://www.mimosa.org/TechCDEV3-2">
   <mim_query_req>
         <recursive_base>
           <recursive_base_params return_class="top_parent" return_fk_tables="0" return="1"/>
            <recursive_base_rows>
              <segment>
                 <filterTYPE column_name="segment_site" equal_value="000003F100000001"/>
              </segment>
            </recursive_base_rows>
            <recursive_base_info>
               <segment_child>
               </segment_child>
            </recursive_base_info>
         </recursive_base>
   </mim_query_req>
</mim_query>
```

EXAMPLE OF SERVER-INTERNAL SQL STATEMENTS GENERATED BY QUERY 4

```
SELECT DISTINCT segment.* FROM segment segment LEFT JOIN segment_child cd on
        segment.segment_site = cd.segment_site  AND
        segment.segment_id = cd.segment_id
                WHERE NOT EXISTS(SELECT * FROM segment_child
                   WHERE segment.segment_site  =
                    segment_child.segment_site
                AND segment.segment_id  = segment_child.segment_id)
        AND segment.segment_site  = '0000042700001000'
```

In this example, the request being sent to the server is asking to return the “top-level” segment parent rows. These are rows in which there are not children (do not have parents). This example shows the strength of leveraging Tech-CDE. Instead of the client being required to generate a complex SQL query, this Tech-CDE query can be sent and the server transform this request into the proper SQL statement and return the XML query results to the client.