

Data Improvements

Data related new features and improvements



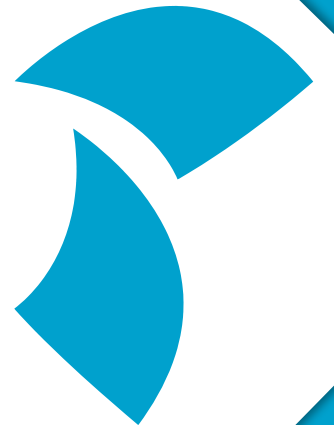
Overview

- Tagged data records
 - Concepts
 - Tagging records with help from Manipula and DataLink API
- Blaise SQL
 - Syntax rules
 - Date and time functions
 - Supported statements; record filters
- Data interface files (*.bdix)
 - Create for specific version
 - Upgrading and downgrading Data Interface files
- Hospital tool



Tagged data records

Storing multiple versions of data records



Tagged data records

- New feature in 5.15
- Main purpose: to support the MMCM
 - Multiple modes can write records simultaneously
 - Prevent data loss
 - Tag table will always be added to the bdix in 5.15
 - Stores copies of data records along with some tag information
 - TagCreated, TagMode, TagIdentityName, TagEvent, TagReason
- Can also be used outside/without the MMCM

Tagged data records

- Tag functionality generally available
 - Reading tagged records
 - Writing tagged records; Tag records yourself
 - Delete tagged records
- Possible use cases
 - Build a record data history and add info to the records
 - Backup data records



Tagged data records

- Tag table versus History table
 - Both tables can store the history of record data
- Difference
 - History table is populated by using database triggers
 - Will be updated on each insert, update and delete of a record
 - Tag table is populated on request
 - Gives you more to control when to update the tag table



Manipula – Writing tagged records

- WriteTaggedData
 - New file method in Manipula
 - Can be used in combination with an update, output and survey data file
- Implementation
 - Existing records will be updated in the main database and the record will also be written to the Tag table
 - If the record does not exist yet in the main database then it will be added, and the record will also be added to the Tag table



Manipula - Reading tagged records

- TaggedDataFile

- Can be used to read tagged data records
- Must always be used together with an input, output or survey data file
 - For example:
 - inputfile F
 - TaggedDataFile T
 - Reading the tagged records of the current record in F
 - F.GetTaggedData(T)
 - Iterate over the tagged records in TaggedDataFile
 - Possibility to restore a tagged record

- Tag properties can only be accessed via reflection

```
message('T.Person.Firstname = ' + T.Person.FirstName + ' T.TagID= ' + [[T.TagID]] + ' TagMode= ' +  
[[T.TagMode]] + ' TagEvent= ' + [[T.TagEvent]] + ' TagReason= ' + [[T.TagReason]] + ' TagIdentityName= ' +  
[[T.TagIdentityName]])
```



Manipula – Deleting tagged records

- DeleteTaggedData
 - Deletes all the tagged records based on the primary key of the current record

Tagging data records – DataLink API

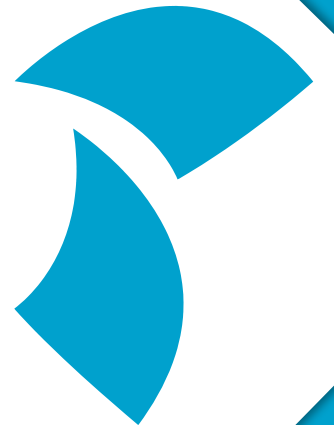
- IDataLink11

- `IDataSet ReadTaggedData(IKey primaryKeyValue, string filter = "");`
- `void WriteTaggedData(IDataRecord dataRecord, string tagMode, string tagEvent, string tagReason, string tagIdentityName);`
- `int DeleteTaggedData(IKey primaryKeyValue);`



Blaise SQL

Concepts and workings



Blaise SQL versus Native SQL

Blaise SQL

- SQL understood by Blaise Data Provider
- Is used by all Blaise applications and DataLink API to access BDIX data
- Can access and filter data that is stored in the binary data stream column
- Can access special column used in bdix files
- Is data source and table structure independent

Native SQL

- Is native to the underlying database that a BDIX is targeting
- Is not limited to the tables which are present in the bdix



Blaise SQL – Syntax rules

- Blaise field names must be fully qualified field names

```
Address.Street = 'Kerkweg'
```

```
Person[1].Name = 'John'
```

```
NrOfPeople > 2 and Town = 'Kerkrade'
```

```
(NrOfPeople > 2) or (Town like 'Ams%' and IntervNo in (1,2))
```

- Names can be delimited by `

```
`NrOfPeople` > 4 and `Town` = 'Kerkrade'
```



Blaise SQL – Syntax rules

- ‘Special’ columns must be surrounded by square brackets []

```
[ValidationStatus] in (0,1)
[FormID] > 1000
[SaveStatus] = 'Completed'
[Mode] in ('CAWI', 'CAPI')
```

- Filtering on null and not null values

```
`Address.Street` is null
Town is not null
[Mode] is null
```

- Names can be surrounded by ` to escape SQL reserved words

```
select `select` from `all`
```



Blaise SQL – Date and time functions

- DateToStr(<date field or column>, '<format string>')
- StrToDate('<date time string>', '<format string>')
- Is translated into native date and time functions used by supported relational databases
- Format String
 - Y, y year
 - M month
 - D, d day
 - H, h hour
 - m minutes
 - S, s seconds
 - F, f milli-seconds



Blaise SQL – DateToStr function

- DateToStr(<date/time field or column>, '<format string>')
- Can be used in select and where clause/ record filters

DateToStr(CMA_StartDate, 'MM') = '07'

DateToStr(CMA_StartDate, 'YYYY-MM-DD') = '2024-03-27'

DateToStr(CMA_StartDate, 'yyyy-MM-dd hh:mm:ss') = '2024-03-27 13:04:06'



Blaise SQL – StrToDate function

- StrToDate('string value', '<format string>')
 - CMA_StartDate = StrToDate('2025-07-25', 'YYYY-MM-DD hh:mm:ss')
- Possible pitfall: time part in database column has a value
CMA_StartDate = StrToDate('2025-07-25', 'YYYY-MM-DD')
will return no results if the time part has a (default) value (for instance 01/01/01)
- Solutions
 - Include time part as well:
CMA_StartDate = StrToDate('2025-07-25 01:01:01', 'YYYY-MM-DD hh:mm:ss')
 - Use >= and < operators in where clause
- CMA_StartDate >= StrToDate('2025-07-25', 'YYYY-MM-DD') and CMA_StartDate < StrToDate('2025-07-26', 'YYYY-MM-DD')



Blaise SQL – Select Statement

- Retrieves Blaise data, including field data that is present in the data stream only
- Special columns can be accessed by surrounding them by []
- Manipula ExecuteQuery and IDataLink.ExecuteQuery

```
select MainSurveyID, SurveyDisplayName, `ID`, CMA_Supervisor,  
       CMA_ForWhom, CMA_Location, CMA_Process.FirstDownloaded.When,  
       [validationstatus], [timecreated]  
from `Launcher`  
where CMA_Supervisor = 'CMA_Sup'
```

- From clause can be <datamodel name>, [TAG] and [HISTORY]



Blaise SQL – Update Statement

- Can update multiple records at once
- Updates also values in the binary data stream
- Where clause can also contain fields that are stored in the data stream
- Manipula ExecuteNonQuery and IDataLink7.ExecuteNonQuery

```
update `Launcher`  
set CMA_ForWhom = 'CMA_Ben'  
where CMA_Supervisor = 'CMA_Sup' and CMA_Location is null
```



Blaise SQL – Insert Statement

- Inserts records by specifying Blaise field names and values
- Inserts one record at a time; values will also be set in the data stream
- Manipula ExecuteNonQuery and IDataLink7.ExecuteNonQuery

```
insert into launcher (MainSurveyID, `ID`, cma_supervisor)  
values ('InstrumentGUID', 'MyID', 'CMA_Sup')
```



Blaise SQL – Delete Statement

- Can delete multiple records at once
- Where clause can contain items that are stored in the data stream
- Manipula ExecuteNonQuery and IDataLink7.ExecuteNonQuery

```
Delete from `Launcher`  
where CMA_Supervisor = 'CMA_Sup' and  
CMA_Location is not null
```

Blaise SQL - Optimizing query performance

- **Good: All items that are in record filter have a column in the database**
 - Corresponding columns will be used in where clause
 - Filter can be applied in the database directly
- **Bad: One or more of the items in record filter do not have a column**
 - Data to be filtered is stored in the data stream only
 - All records must be read because we cannot filter the data in the database itself!
 - Requested record data will be loaded into a ADO.NET DataTable
 - DataTable will be filtered with the specified filter using by using a ADO.NET DataView
 - Filtered data in DataView will be returned



Optimizing query performance

Data Partition Type	Filtering key fields	Filtering non-key fields	Comment
Stream	😊	😞	Key fields have dedicated columns, non-key fields are stored in data stream
Flat, Blocks	😊	😊	Every field has its own column
Flat, No blocks	😊	😊	Every field has its own column
In Depth	😊	😐	Non-key fields must be filtered by using the in-depth data table; can have many rows
Generic Stream	😞	😞	No way to filter based on individual field values; key values are stored as a concatenating string
Generic In Depth	😐	😐	Fields must be filtered by using the in-depth data table; can have many rows
Single Table	😊	😊	Every field has its own column



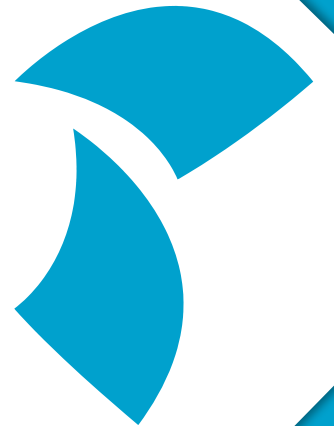
Optimizing query performance

- Recommendation
 - Investigate which record filters that you want to use
 - Add a flat data table to your bdix that contains columns for items that don't have a dedicated column in the database or are stored in a in-depth way only
 - Create indexes on columns to optimize performance even further
 - Blaise will use these columns automatically when they are present
 - Populate flat table when the other tables have already data
 - Will be done by automatically by Hospital 5.14 an later



Data Interface files

Custom create, upgrade and downgrade

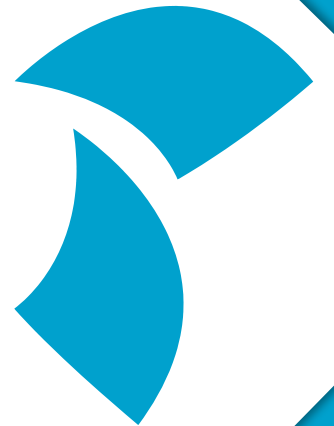


Data Interface new features

- Create data interface for a specific Blaise version
- Upgrading
 - Adds columns and tables if necessary
- Downgrading
 - Content of the bdix is used; only tables and columns in bdix will be accessed
 - Additional tables and columns might exist in database
- Available options are determined by Blaise bmix version

Hospital Tool

Validate and repair



Hospital Tool

- Main purposes
 - Make data consistent again
 - Try to recover data that has become corrupted
- Contains
 - BDIX validation check
 - Data consistency check
 - Can report, export or repair inconsistencies



Hospital Tool

- Recover options
 - Recovery based on field names and values in data stream
 - Existing fields will be recovered
 - Non stream data partition types
 - Take data stream and make data in data tables consistent
 - Take data in data tables and make data in data stream consistent
- Export possibility for incompatible data streams to XML

