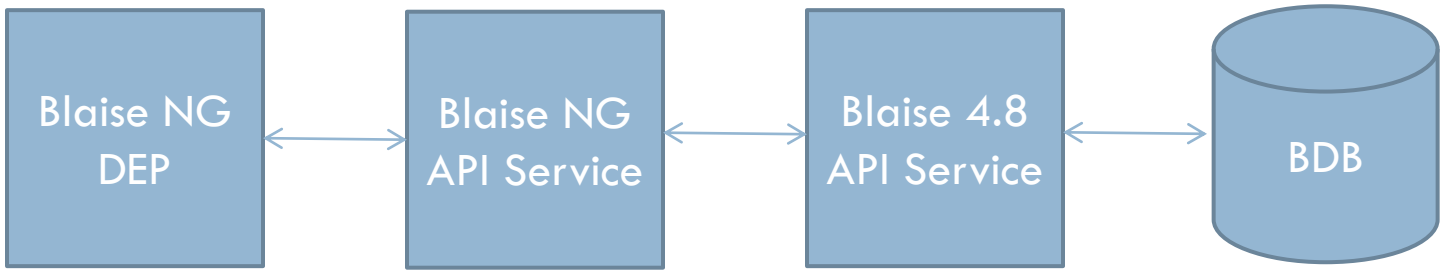
BLAISE NG
DATA MANAGEMENT

Phase 1 CTP: Data access

- Use of 4.x data files
  - BDB's
  - BOI support only if Blaise 4.8 installed
- Access to data files exclusively through Blaise NG API service and Blaise 4.8 API service

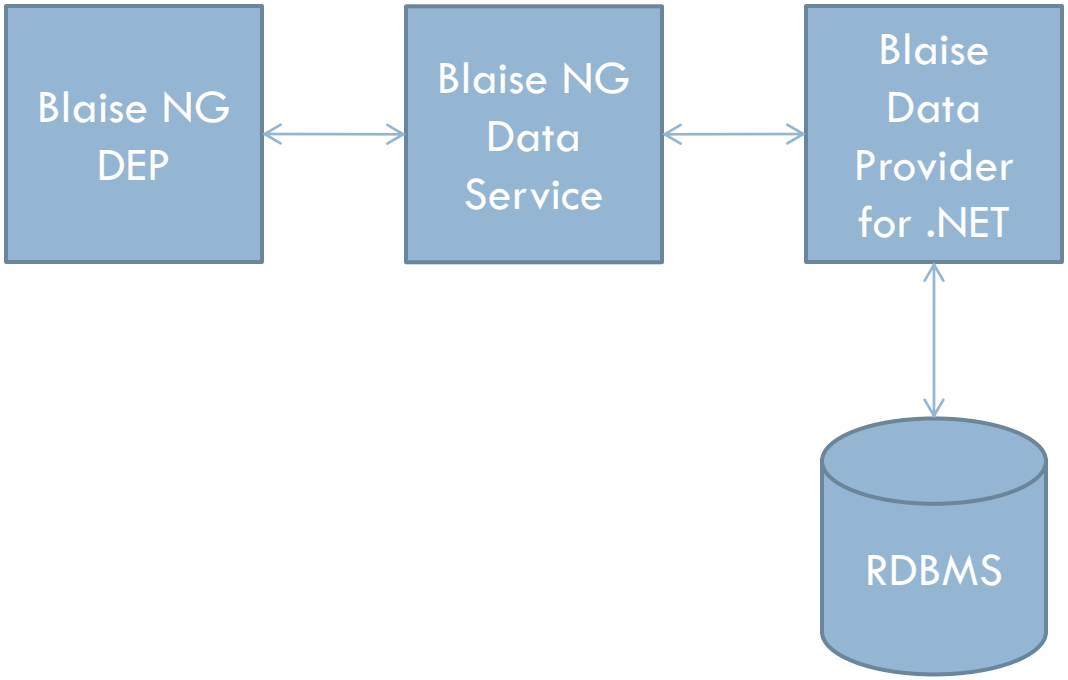
Phase1(CTP) : Database access



Phase 2: Data access

- Replace the 4.x Blaise database (BDB) files
- Add support for most popular relational databases
- Be compatible with the 4.x search capabilities
  - Implement trigram keys
  - Add lookup functionality
- Change the data access process
  - Phase out data access through the 4.8 API Service
  - Use Blaise Data Provider for .NET instead

Phase2: Database access



Ingredients to make this work

The Blaise NG Database

- Extension BDBX
- Default data storage option in NG
- Relational database format; SQLite based
  - Lightweight SQL database
  - Open source; C++ code is available
  - Transactional, zero-administration, serverless, self-contained
  - Unicode support; Multi-platform
  - SQLite .NET Data Provider

Demo: Automatic creation of required data files by running the DEP

Blaise Data Interface files

- Extension BDIX
- Main data file in Blaise NG; not the BDBX
- Uses the same concepts as a BOI file
  - Contains logical information; no data
    - ConnectionString
      - Database specific connection string
    - Data Provider and Data Source information
    - Table definitions and their structure
    - Data Model information

Blaise Data Interface files

- Supported providers and drivers
  - .NET Framework Data Providers
    - Managed providers
      - .NET Framework Data Provider for SQL Server
      - Oracle Data Provider for .NET
    - Preferred and most direct way to connect
  - OLE DB Data Providers
    - Access through Microsoft .NET Data Provider for OLE DB
      - Microsoft Jet OLE DB Provider
  - ODBC drivers
    - Access through Microsoft .NET Data Provider for ODBC
      - MySQL Connector ODBC

Blaise Data Interface files

- Initially supported data sources
  - Blaise database files (*.bdbx)
  - Microsoft SQL Server
  - Microsoft Access (*.mdb; *.accdb)
  - Oracle
  - MySQL
  - SQLite
  - Text files

Demo: Create a data interface based on an existing database table

Blaise NG Data Storage

- We plan to include the following storage features
  - Non-generic / generic storage
  - Several storage structures
    - Stream
    - Type specific data columns
  - Meta and data versioning
    - Special versioning columns can be added optionally
      - DataModelKey
      - BeginStamp and EndStamp

Demo: Create a data interface and Blaise database file via Wizard

What’s in the background

Components Overview

Blaise Data Provider for .NET

- BlaiseConnection
- BlaiseCommand
- BlaiseDataReader
- BlaiseDataAdapter
- SQLComposer (Not Public)

Blaise SQL Parser

Builds Abstract Syntax Tree for a Blaise SQL statement

Data Access Layer

Classes for Abstract Data Access

Object Oriented SQL Statement Builder

Supported Databases

- Oracle
- SQL Server
- MS Access
- SQLite
- MySQL
- …

Blaise Data Provider for .NET

- Used by Blaise client applications to access data
- Implements required .NET Data Provider interfaces
- Can be used in order to access
  - Blaise database files (bdbx)
  - Relational databases
- 'Talks' Blaise SQL
  - SQL using Blaise syntax rules and reserved words

Blaise Data Provider for .NET objects

- BlaiseConnection
  - Opens connection to a data source
  - ConnectionString typically contains the data interface file to open
- BlaiseCommand
  - Executes command against the data souce
- BlaiseDataReader
  - Fast forward readonly cursor
- BlaiseDataAdapter
  - Fills and updates data sets

Using Blaise Data Provider for .NET

```
// Create a BlaiseConnection, set the connection string and open the connection
BlaiseConnection connection = new BlaiseConnection(@"Data Source=d:\blaiseng\all\all.bdix");
connection.Open();

// Create a BlaiseCommand in order to query data
BlaiseCommand command = new BlaiseCommand();
command.Connection = connection;
command.CommandText = "select intervno, household.person[1].name from all";

// Create an ADO.NET DataTable tat will ve
DataTable dataTable = new DataTable();

// Use a BlaiseDataAdapter object to fill the datatable
BlaiseDataAdapter da = new BlaiseDataAdapter(command);
da.Fill(dataTable);

// Close the connection
connection.Close();
```

Blaise SQL examples

```
// Create a BlaiseConnection, set the connection string and open the connection
BlaiseConnection connection = new BlaiseConnection(@"Data Source=d:\blaiseng\all\all.bdix");
connection.Open();

// Create a BlaiseCommand in order to query data
BlaiseCommand command = new BlaiseCommand();
command.Connection = connection;

// Blaise SQL for selecting data
command.CommandText = "select intervno, household.person[1].name from all";
command.CommandText = "select [FORMSTATUS], [FORMID], intervno from all";
command.CommandText = "select intervno from all where town = 'heerlen'";

// Blaise SQL for doing key searches
command.CommandText = "select intervno, town from primary where searchcondition = 'he'";
command.CommandText = "select name, state, town from sk_name where searchcondition = 'fra'";
```

Blaise SQL Parser

- Blaise SQL will be parsed by the Blaise SQL Parser
- ANTLR based
- Result of the parsing process is a statement dependent Abstract Syntax Tree
- SQLComposer reads the AST and builds a database specific SQL statement
  - By using the Data Access Layer (DAL)
  - By taking into account
    - Table structures and storage structure as defined in bdix
    - The native SQL syntax of the database

Data Access Layer (DAL)

- Interacts with the data source
- Builds data source's native SQL
- Client apps of DAL uses abstract base classes
  - ConnectionBase
  - SelectStatementBase
  - UpdateStatementBase
  - …
- OO approach for building SQL statements

Data Access Layer (DAL)

```
// Create abstract connection
ConnectionBase connection = new MySqlConnection("conn string");

// Create select statement base on connection
SelectStatementBase ss = connection.CreateSelect();

// Add some items to the select, from, where and order by clause
ss.Select.Add("intervno");
ss.Select.Add("street");
ss.From.Add("all");
ss.Where.And(new ConditionEqual(new Identifier("town"),
    new Constant("kerkrade")));
ss.OrderBy.Add("nrofpeople");

// Get the native SQL (in this case MySQL)
string nativeSQL = ss.CommandText;

// Execute the statement and create an ADO DataTable
connection.ExecuteDataTable(ss.CreateCommand());
```