4D FOR FLEX V1.1 MANUAL

INSTALLATION

The whole library has been split into three folders, each containing a package (.swc file):

- **Flex4D_SQL** is the main folder: it contains the Flex4D_SQL.swc package. This package contains everything needed to connect to a 4D SQL Server, especially the SQLService Class (see below).

- **Flex4D_Controls** contains the Flex4D_Controls.swc packages. It brings the UI Controls (Custom DataGrid , Navigation Button, etc.) to the library.

- **Flex4D_Components** contains the Flex4D_Components.swc package. It provides an interface for using the trace facility and building an utility similar to AIR TraceConsole (see the corresponding section).

INTRODUCTION

USING FLEX AS A CLIENT FOR 4D SERVER

Despite the quality of the Flex framework, accessing data from a server remains a challenge.

Since the 2003 version, 4D can behave as an application Server through its built-in HTTP Server. It can serve HTML pages, Flex applications (SWF). With just a single click, a developer can publish a method through HTTP or SOAP. The use of Flex components like mx:HTTPService or mx:WebService therefore makes data exchange easy between Flex and 4D. Things are getting even easier with Flex Builder 3 and its WSDL wizard (Import Web Service (WSDL), etc.).

DIRECT ACCESS THROUGH SQL

So far, programming a Flex Client for a 4D database has been possible with recent versions of 4D, however the HTTP approach presents several drawbacks:

- You need to program an API (HTTP or SOAP) on the server side and to serialize the data exchanged, often in XML.
- As a consequence, performance suffers from large amounts of data being exchanged.
- It is not easy to send updates to the database; you have to write many functions in order to do so.
- Since server-side coding is required, you have to coordinate the work on client and server.

With the 4D v11 version, it is now possible to access the SQL Server externally and to execute SQL queries directly on the engine, without requiring an ODBC driver. This requires using a proprietary SQL protocol.

Therefore, we implemented this SQL protocol in ActionScript for Flex. It offers an efficient new way of communication between Flex and 4D: no server-side code required, fast communication, no XML, plain SQL queries, etc.

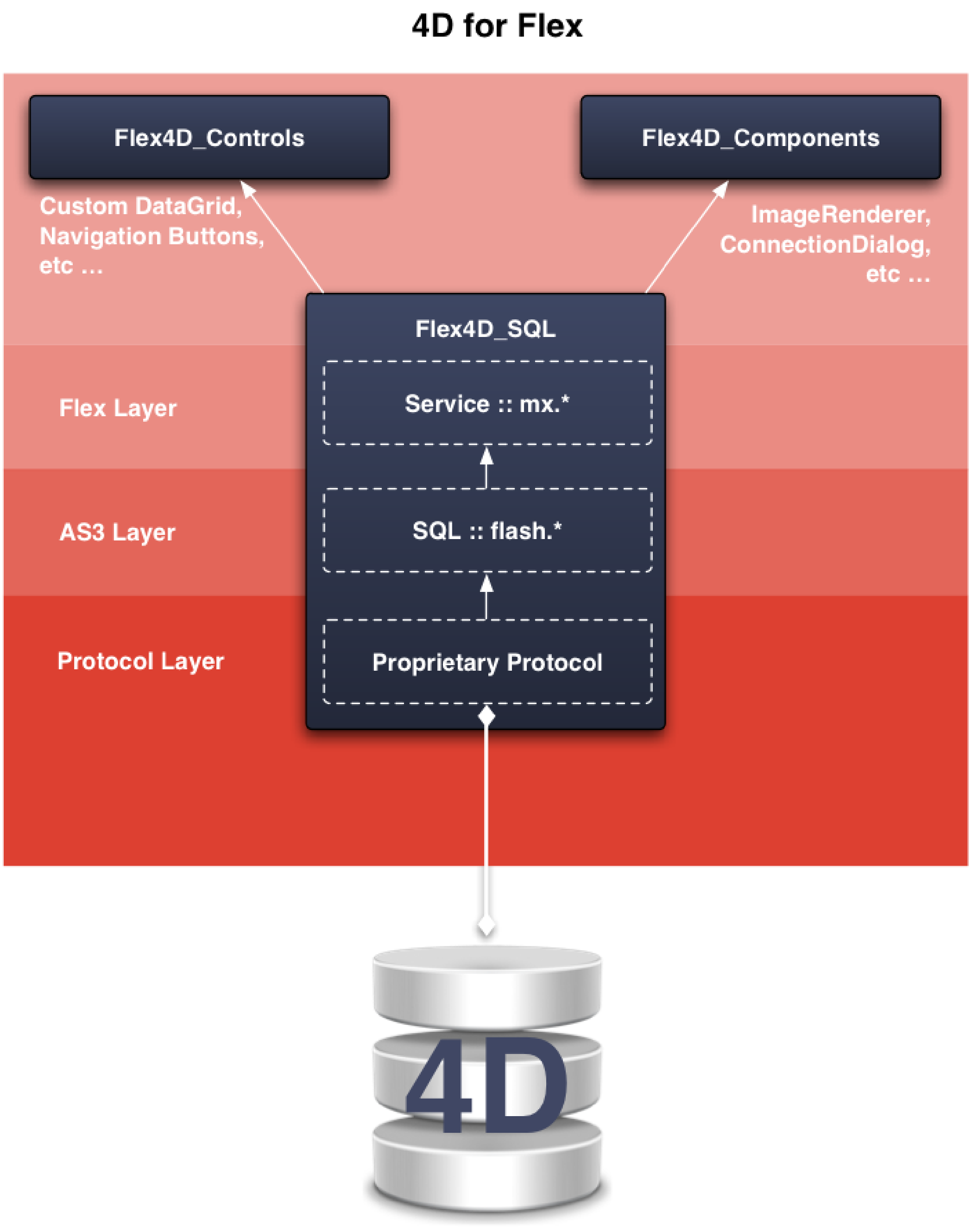
The SQL approach is very powerful but presents some limitations as well:

- A Flex developer must know the relational model of the database in order to write SQL queries.
- Contrary to the disconnected mode used by XML over HTTP or SOAP, a Flex client works in a connected mode. This can make the communication more fragile on WAN or Wi-Fi networks. An automatic reconnection mechanism has been built into the library to minimize this issue (*).
- The SQL Server listens to a specific TCP port, different from the HTTP port 80. You may need to provide access to this port on a firewall. Due to security restriction of the Flash Player, you also need to explicitly allow socket connection to this port.

(*) This mechanism is not activated in version 1.0.

4D FOR FLEX ARCHITECTURE

The 4D for Flex component is composed of several layers. As you move up in the hierarchy, the complexity is hidden.



4D for Flex architecture

- **Protocol Layer** is private. It allows a Flex client to communicate with the SQL Server using a proprietary SQL protocol.

- **AS3 Layer** is intended to be public and used with ActionScript 3 with no dependency on the Flex framework (*mx* package). Flash CS3 developers can then use 4D for Flex to execute SQL queries on 4D SQL Server. However, it does require some coding effort. In the current version (1.0) of the library, we discourage using this API since it will change in upcoming versions. This package is very similar to the *flash.data* package provided by Adobe in AIR to access the local SQLite databases.
- **Flex Layer** contains a MXML component for Flex developers called **SQLService**. It provides a very simple way to execute SQL queries in the 4D SQL Server and is recommended for first-time developers. The SQLService API is very similar to other data services used in Flex: HTTPService or WebService for instance.

These three layers are grouped in a single **Flex4D_SQL** package which also contains a trace utility to log communication traffic between a client and a 4D SQL server. This package is the basic building block of 4D for Flex.

You can also use optional packages:

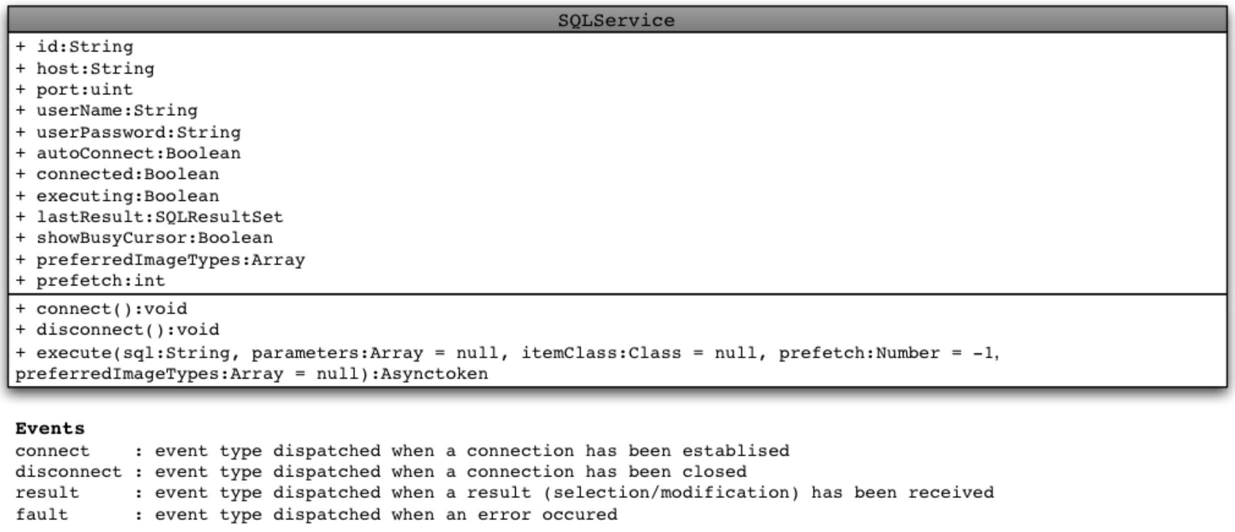
- The **Flex4D_Controls** package provides several cool features to the Flex developer: a custom DataGrid and a set of navigation buttons which make it simpler to interact with the 4D selection of records, etc.
- The **Flex4D_Components** package provides ready-to-use components: ImageRenderer and BooleanRenderer for a dataGrid item, ConnectionDialog, Language selector, etc.

SQLSERVICE CLASS

For a Flex developer, this class is the core of 4D for Flex and the easiest one to use for first-time developers.

CLASS DIAGRAM

The SQLService can be used with the MXML language. Here is the diagram for this class:



```
SQLService
+ id:String
+ host:String
+ port:uint
+ userName:String
+ userPassword:String
+ autoConnect:Boolean
+ connected:Boolean
+ executing:Boolean
+ lastResult:SQLResultSet
+ showBusyCursor:Boolean
+ preferredImageTypes:Array
+ prefetch:int
+ connect():void
+ disconnect():void
+ execute(sql:String, parameters:Array = null, itemClass:Class = null, prefetch:Number = -1,
preferredImageTypes:Array = null):Asynctoken

Events
connect    : event type dispatched when a connection has been establised
disconnect : event type dispatched when a connection has been closed
result     : event type dispatched when a result (selection/modification) has been received
fault      : event type dispatched when an error occured
```

Let's describe this class.

PROPERTIES

- **id**: String. You specify the name that will be used in AS3 code to refer to the component. It corresponds to an ActionScript variable name.
- **host**: String. The IP address or the name of the host where the SQL Server can be reached.
  - For execution in a browser:

    This is not mandatory. If no host is specified, the component will try to connect to the host which served the containing SWF. If the SQL Server address is different from the address of the Web Server which serves the SWF, you must specify the SQL Server address explicitly.
  - For execution as an AIR application:

    You must provide a host.
- **port**: Number. By default, 4D SQL Server listens to the port 1919. This can be modified in the database preferences. If no port is specified, the component will try to connect on the default port.
- **userName**: String. Normally, you must authenticate to the SQL Server in order to execute queries. If the database doesn't use any authentication (which is unwise), you can skip this attribute as well as the following (userPassword).
- **userPassword**: String. Here you specify the password matching the userName.

  See the Security section for more information on how to authenticate from Flex.
- **autoConnect**: Boolean. This attribute is not mandatory and by default is set to False. If explicitly set to True, the component will automatically try to connect to the SQL Server when instantiated.
- **connected**: Boolean. Read-only. Indicates whether or not the component is currently connected to the SQL Server.
- **executing**: Boolean. True if an operation is being executed on the SQL Server. False otherwise.
- **lastResult**: SQLResultSet. Read-only. Contains the last ResultSet received from the SQL Server.
- **showBusyCursor**: Boolean. This attribute is not mandatory and by default is set to True. If set to True, a busy cursor is displayed while a service is executing.
- **preferredImageTypes**: Array. Declare the list of image types accepted by the Flex client. The SQL Server tries to send the image using the first matching type.

  E.g.: `preferredImageTypes="['.png', '.jpeg', '.gif']"`

In the 4D database, if an image is stored in one of the specified formats, it will be returned to the client as is. If an image is stored in another format, it will automatically be converted to the first format of the list, i.e. *PNG* in the example. By default, if **preferredImageTypes** is not specified, images are returned in a default format.

In this example, if 4D Server has a PNG version of the image, it will send it. If not, it will try to return a JPEG version and so on...

If no format is specified and the image is available in several formats on the Server, the first image stored will be returned. So, you'd better always specified a preferred type.

- **prefetch**: int. The SQL Server returns the selected rows in paginated mode. This attribute declares the number of rows in a page. The default value is 100.

PROPERTIES GRID

Property name	AS3 Type	Accessibility	Mandatory	Default value	Description
host	String	read / write	yes	hosting server / 127.0.0.1	The 4D SQL Server address.
port	Number	read / write	yes	19812	The 4D SQL Server port.
userName	String	read / write	no		The 4D user name. If the 4D database designer has no password, this property is ignored.
userPassword	String	read / write	no		The 4D user password. If the 4D database designer has no password, this property is ignored.
autoConnect	Boolean	read / write	no	false	Set to *true* if you want to connect automatically. Set to *false*, if you want to connect manually.
connected	Boolean	read	no	false	*true* if the connection is opened on a 4D SQL Server, *false* otherwise.
executing	Boolean	read	no	false	*true* if an operation is being executed on a 4D SQL Server, *false* otherwise.
lastResult	SQLResultSet	read	no	null	The last result set returned by a SQL SELECT query.
showBusyCursor	Boolean	read / write	no	true	Set to *true* if you want to show a busy cursor when an operation is being executed on a 4D SQL Server. Set to *false* otherwise

EXAMPLE:

You can declare an SQLService with the following code:

```
<fourD:SQLService
```

```
            id = "fourDSQLService"
            host = ""
            userName = "sqlUser"
            userPassword = "..."
            autoConnect="true"
        />
```

And you can refer to this SQLService later:

```
        <mx:Button label="Connect" click="fourdSQLService.connect()"/>
```

or:

```
        <mx:Button label="Connect" click="sqlConnect()"/>
```

with the function sqlConnect():

```
        private function sqlConnect():void{
            fourDSQLService.connect()
        }
```

METHODS

- **SQLService()**

  This is the constructor method.

- **connect**():void

  Tries to connect to the specified host and to authenticate using the provided user and password values. If the connection succeeds, a connect event is triggered.

- **disconnect**():void

  Disconnects from the SQL Server. No more queries can be sent after calling this method. A disconnect event is triggered.

- **execute** (sql:String, parameters:Array = null, itemClass:Class = null, prefetch:Number = -1, preferredImageTypes:Array = null):AsyncToken

Asks the SQL Server to execute the SQL query 'sql'. The other parameters are not mandatory. See examples below.

- **iterate**(resultSet:SQLResultSet, iteratedFunction:Function):void

  Since the result of a query is paginated, if the number of selected records exceeds the prefetch value, it is not possible to iterate through the SQLResultSet as usual in an ArrayCollection. That's why the SQLService provides a specific iterate function. In the first parameter, you pass the SQLResultSet returned by the SQL Server and in the second parameter, you pass the name of a function that will be called for each record of the **complete** selection. In other words, the iterate() function will navigate through the whole selection and automatically fetch new pages when needed.

  See example below. **Not available in version 1.0**

EXECUTE() METHOD

PARAMETERS

See the "Using dynamic statements paragraph" for using parameters in queries.

ITEMCLASS

itemClass:Class = null

itemClass is part of the roadmap for future version

The following is copied from the Adobe doc:

Indicates a class (data type) that is used for each row returned as a result of the statement's execution.

By default, each row returned by a SELECT statement is created as an Object instance, with the result set's column names as the names of the properties of the object, and the value of each column as the value of its associated property.

By specifying a class for the itemClass property, each row returned by a SELECT statement executed by this SQLStatement instance is created as an instance of the designated class. Each property of the itemClass instance is assigned the value from the column with the same name as the property.

Any class assigned to this property must have a constructor that does not require any parameters. In addition, the class must have a single property for each column returned by the SELECT statement. It is considered an error if a column in the SELECT list does not have a matching property name in the itemClass class.

See the Adobe doc for an example:

http://livedocs.adobe.com/flex/3/html/help.html?content=SQL_08.html

Instead of writing:

```
var item:Object = resultSet.getItemAt(i);

var dept:String = item["[dept]"];

var id:String = item["[id]"];
```

where the item is not typed, you can use itemClass.

First, define a class with a public property for each column of the SELECT query. If the query is:

```
SELECT lastName, firstName, address, zipCode, City FROM contacts
```

You must write:

```
public class Contact
        {
                public var lastName:String;
                public var firstName:String;
                public var address:String;
                public var zipCode:String;
                public var city:String;
        }
```

Warning : in a true example, you should use accessor methods and not public variables. We use this for keeping the code as concise as possible.

Then perform the SELECT attaching itemClass

```
var sql:String = "SELECT lastName, firstName, address, zipCode, city FROM contacts";

fourDSQLService.execute(sql, Contact);
```

Finally, in the eventHandler result, read the first item:

```
var contact:Contact;

contact = event.result.getItemAt(0);
```

and access each property, for instance to display the values in text fields:

```
this.lastNameField.text= contact.lastName;

this.firstNameField.text=contact.firstName;

this.addressField.text= contact.address;

this.zipCodeField.text= contact.zipCode;

this.cityField.text= contact.city;
```

PREFETCH

prefetch:Number = -1

When the statement's text property is a SELECT statement, this value indicates how many rows are returned at one time by the execution of the statement.

If specified and different from -1, this value is used for the execution of the statement independently of the SQLService.prefetch property value.

PREFERREDIMAGETYPES

preferredImageTypes:Array. Declares the list of image types accepted by the Flex client. The SQL Server tries to send the image in the first matching type.

Default value is null so that SQLService.preferredImageTypes is used.

If a value other than null is passed, this value is used for the execution of the statement independently of the SQLService.preferredImageTypes property value.

ASYNCTOKEN

See the Advanced topics/Using AsyncToken paragraph.

EVENTS

- **connect**

  Triggered when the component is connected to an SQL Server.

- **disconnect**

  Triggered when the component is disconnected from an SQL Server.

- **fault**

  Triggered when something wrong happens while communicating with the SQL Server.

- **result**

  Triggered when a response is returned by the SQLServer.

See examples below.

EXAMPLES:

DECLARATION OF A SQLSERVICE

```
<fourD:SQLService
      id = "fourDSQLService"
      host = ""
      userName = "Administrator"
      autoConnect="true"

      result = "resultHandler(event)"
      fault = "faultHandler(event)"

      showBusyCursor="false"
      preferredImageTypes=['.png', '.jpeg', '.gif'];
/>
```

SQLService automatically tries to connect to the host that served the SWF, on the SQL Server default port, and to login as 'Administrator' with no password.

In case of problems, the faultHandler() method is called.

After receiving the result of a query, resultHandler() is called.

A busy cursor is not displayed when a network operation is being executed.

If the result of the query contains images, the SQL Server returns images which are encoded in PNG, JPEG or GIF.

Note

a SQLService is declared in a namespace identified by http://www.4d.com/2007/mxml that has to be declared as a property of your component. In an mx:Application component, it would be declared as: `<mx:Application>` tag:

```
E.g.:
<mx:Application
      xmlns:fourD="http://www.4d.com/2007/mxml"
      ...
>
```

EXECUTION OF QUERIES

Once connected, you can send queries to the database:

```
var sql:String = "SELECT id, firstName, surname, address, zipCode,
city FROM Contact";

fourDSQLService.execute(sql);
```

This code will ask the SQL Server to execute an SQL query. See the next paragraph to find out how to get the resulting rows.

RETRIEVING RESULTS

Suppose we have declared an eventListener resultHandler() for the result:

```
result = "resultHandler(event)"
```

We can write the resultHandler method as follows:

```
private function resultHandler(event:ResultEvent):void
```

```
    {
          var resultSet:SQLResultSet;
          resultSet = event.result as SQLResultSet;
          recordCountLabel.text=String(_resultSet.nbRecords)+" record(s)
    found";
    }
```

First, we cast event.result as an SQLResultSet Object which implements the ICollectionView interface and thus can be used directly as a dataProvider for a DataGrid for example.

Then we can use SQLResultSet properties (see the HTML doc) among them:

- nbLoadedRecords : int

  [read-only] The current number of records loaded locally from the current selection.

- nbRecords : int

  [read-only] The number of records in this result set.

- prefetch : int

  [read-only] The number of records retrieved in a page.

So, in the code above, we assign the number of records in this result set to the text property of a label. The nbRecords property can be very different from nbLoadedRecords which represents (*only*) the records currently loaded from the Server.

Suppose your query selects 500 records and the prefetch property is set to 100. After the query:

- nbRecords is 500
- nbLoadedRecords is 100

Suppose you now fetch 100 new records:

- nbRecords will still be 500
- nbLoadedRecords will now be 200

USING DYNAMIC STATEMENTS

In this example, we use dynamic SQL. In the statement the requested zipCode value is replaced by a question mark.

```
var sql:String = "SELECT firstName, surname, zipCode, city FROM
Contact WHERE zipCode=?";
```

Parameters are indexed by their position in the statement.

Then, we declare a new array with a string value:

```
var parameters:Array = new Array();
parameters.push("50530");
```

This array contains the values that will replace the question marks at execution. Value indices must match the parameter positions.

We can now execute our query passing the array of parameters as the second argument of the execute() method.

```
fourDSQLService.execute(sql, parameters);
```

ITERATING THOUGH A SQLRESULTSET

iterate() is part of the roadmap for future version

FAULT EVENTS

What happens if something goes wrong? If the SQL Server throws an error?

A Fault event will be raised; you must declare an eventListener for catching this important event.

When the listener is called, you get all the information needed in the event parameter:

```
private function faultHandler(event:FaultEvent):void
{
        var errorCode:String = event.fault.faultCode;
        var errorString:String = event.fault.faultString);
        Alert.show(errorString,errorCode);
}
```

USING SQLRESULTSET

SQLRESULTSET CLASS

- **complete**: Boolean. Read-only. Indicates whether the last query has been completely executed.

  We have received all the rows of the last query.
- **refreshQuery**(): calling this method you can execute again the last query performed to SQL Server and so , for instance, automatically refresh a bound DataGrid

DISPLAYING RESULTS IN A DATAGRID

First declare the DataGrid as follows:

```
<fourD:DataGrid
        width="95%"
        height="180"
        id="contacts_dg"

        editable="false"
        textAlign="center"
/>
```

Then bind the dataProvider property to the SQLResultSet sent by the SQLServer:

```
        dataProvider="{_resultSet}"
```

Declare _resultSet as a bindable variable:

```
[Bindable]
private var _resultSet:SQLResultSet;
```

And fill _resultSet when the result is received from the Server:

```
private function resultHandler(event:ResultEvent):void
{
	_resultSet = event.result as SQLResultSet;
}
```

FIELDS PROPERTY

We can get information on the fields composing a row by using the fields property. For instance, if we want to get the name of the first column of the row, we can write:

```
_resultSet.fields[0].name
```

OTHERS

You can obtain several items of information about the rows and columns by using the SQLResultSetMetaData object. See a complete description in the HTML documentation. For instance, if we want to know whether the second field of the row is updatable, we write:

```
var contact_metadata:SQLResultSetMetaData=_resultSet.metaData;

if (contact_metadata.isFieldFromIndexUpdateable(1))
	{
		// do something
	}
```

SQLSec

SQLSECUREDSERVICE CLASS

This feature is only available in v1.1

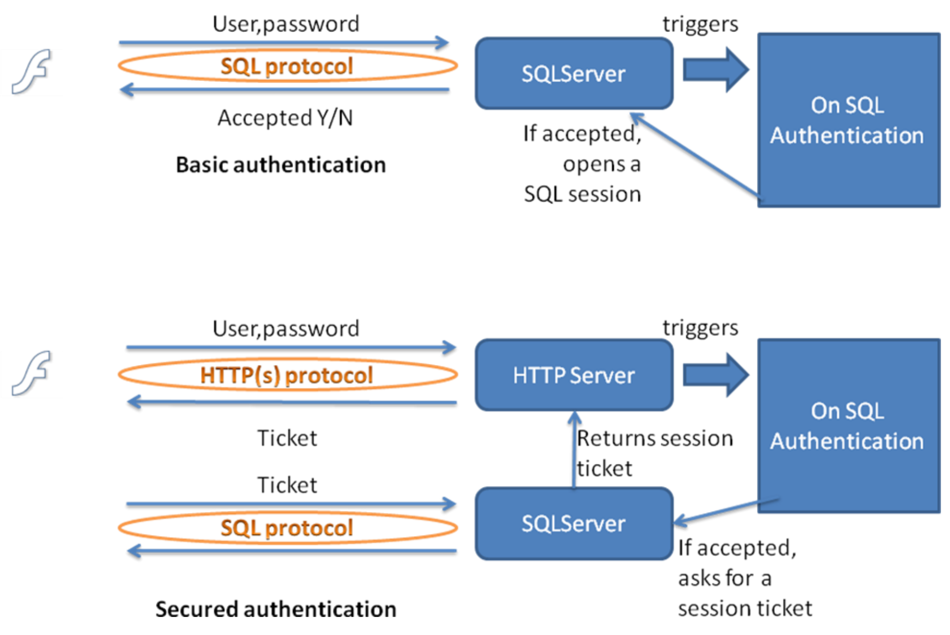
Security of transactions on a public network is a major concern. 4D for Flex allows creating a secured session by avoiding the sending of clear user-passwords on the network. In addition, this information can be virtual and the 4D Developer can match it to existing users on the server side. Finally, the data can be digitally signed to exclude the possibility of an external attack. All of this can only be done due to the exceptional integration of 4D Server functionalities.

SECURED AUTHENTICATION

The connect() method in the SQLService class attempts to connect directly to a 4D SQL Server. This connection scheme can raise some security problems as binary sockets in ActionScript 3.0 are not encrypted and therefore transfer login information in clear text.

To prevent connections from being hacked, a specific 4D connection scheme has been created. Here are the basic steps:

- A client makes an SSL-secured authentication through the 4D built-in HTTP Server;
- If authentication succeeds, an identified token is given back to the client;
- The client logs onto a 4D SQL Server with this token.



Two authentication schemes

This mechanism allows the user names and passwords to be SSL-encrypted and prevents client and server communications from being hijacked during authentication. Of course, this secured authentication process requires the 4D built-in HTTP Server to be running.

USING SQLSECUREDSERVICE CLASS

This mechanism is employed when using the SQLSecuredService class which basically inherits the SQLService class but provides a secured authentication transaction.

Example of minimal declaration of an SQLSecuredService:

```
<fourD:SQLSecuredService
        id = "fourDSQLService"
        httpUseSSL="true"
        userName = "Administrator"
        autoConnect="true"

        result = "resultHandler(event)"
        fault = "faultHandler(event)"

        showBusyCursor="false"
        fourDSQLService.preferredImageTypes=['.png', '.jpeg', '.gif'];
/>
```

In this example, we assume:

- that both the SQL Server and the HTTP Server are running in the same domain: the one which initially served the SWF;
- that sqlPort gets its default value (19812) and httpPort its default value (443) for SSL communication.

We specify that we want the component to attempt an SSL secured authentication at startup using the "Administrator" username and no password.

Here is another example explicitly specifying all parameters needed for the secured authentication:

```
<fourD:SQLSecuredService
id = "fourDSQLService"
host = "sql.myDomain.com"
port = "19812"
httpHost ="www.myDomain.com"
httpPort="8443"
httpUseSSL="true"
userName = "Administrator"
autoConnect="true"
```

```
result = "resultHandler(event)"
fault = "faultHandler(event)"

showBusyCursor="false"
fourDSQLService.preferredImageTypes=['.png', '.jpeg', '.gif'];
/>
```

All the developer needs is to:

- provide the correct parameters to the component declaration;
- (optionally) write some code server-side in the On SQL Authentication database method in order to verify login information and to accept (or refuse) the incoming query.

Once the connection has been established in a secured mode, the data sent by the binary protocol are not sent in an SSL tunnel.

EXAMPLES OF HTTP EXCHANGES

REQUEST

```
POST /4DSQLAUTH/ HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; fr; rv:1.9.0.5) Gecko/2008120121 Firefox/3.0.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Referer: http://localhost:8080/flex/testSQL.swf
Content-type: application/x-www-form-urlencoded
SQL4D-UserPassword: sql
SQL4D-UserName: sqlUSer
Content-length: 1
```

RESPONSE

```
HTTP/1.1 200 OK
Server: 4D_v11_SQL/11.3.0
Date: Tue, 06 Jan 2009 11:10:19 GMT
Content-Length: 152
Content-Type: text/html;charset=UTF-8
Expires: Tue, 06 Jan 2009 11:10:19 GMT
Pragma: no-cache
OK - Authentication successful.
SQL4D-Salt:176179138160329712814571812921521788231 62
SQL4D-SessionTicket:82832462322823395107204160148173391201791 54
```

ADVANCED TOPICS

Although with the features we already saw, you are able to:

- send queries to the SQL Server
- retrieve results and
- display them in a dataGrid for instance,

there are still several advanced topics that we'll cover now.

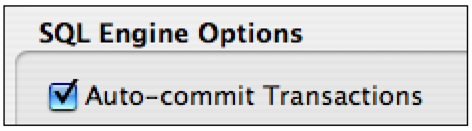
USING TRANSACTIONS

Transactions can be managed automatically by the Server or manually through SQL commands.

AUTOCOMMIT

"Auto-commit mode means that when a statement is completed, the commit method is automatically called on that statement. Auto-commit in effect makes every SQL statement a transaction. The commit occurs when the statement completes or the next statement is executed, whichever comes first. In the case of a statement returning a ResultSet, the statement completes when the last row of the ResultSet has been retrieved or the ResultSet has been explicitly closed."

A preference in the SQL group of 4D Server tells the SQL Server to automatically wrap each query in a transaction. If the query succeeds, the transaction is committed; otherwise, a rollback is done. This mechanism is also known as "Implicit Transaction" in other environments.


SQL Engine Options

✓ Auto-commit Transactions

If this option is selected (recommended), you don't have to manage transactions yourself. Otherwise, you must open and close transactions using SQL commands: START TRANSACTION, SQL COMMIT, SQL ROLLBACK.

See the 4D_v11_SQL_Reference.pdf manual.

LOCAL CHANGES

You can use the methods of SQLResultSet to change data in the ResultSet:

- deleteRecordAt(index:int):void
- insertRecord(record:Object):void

- updateRecordAt(index:int, fieldIndex:int, value:*):void

When changes are sent to the Server, they are automatically executed in a transaction, no matter what value has been set for the Auto-commit Transactions preference.

If the SQLResultSet.autoSubmitChanges property is True, each change is automatically and immediately sent to the Server. You cannot cancel a change.

If the value of SQLResultSet.autoSubmitChanges is False, you must explicitly call:

```
SQLResultSet.submitChanges()
```

to send the modifications of the ResultSet to the 4D SQL database.

If you want to cancel all local pending modifications manually, you can call SQLResultSet.cancelChanges().

As said before, 4D for Flex automatically submits changes in a transaction. So, if an error occurs Server-side, all the changes submitted in the transaction are canceled on the Server. This prevents you from being in an unknown state where partial changes are done while others have failed. In this case, you still have the changes in SQLResultSet, so you can call SQLResultSet.cancelChanges() if you don't want to try to submit them to the Server again.

CRUD

When talking about data services, an acronym that is frequently encountered in different technologies is CRUD. It means:

- C: Create a record on the Server
- R: Read (or Retrieve) a record from the Server
- U: Update a record on the Server
- D: Delete a record from the Server

So, what about 4D for Flex and CRUD?

The first benefit from using SQLService is that you can accomplish all operations directly through SQL queries. No need to use a special API for that.

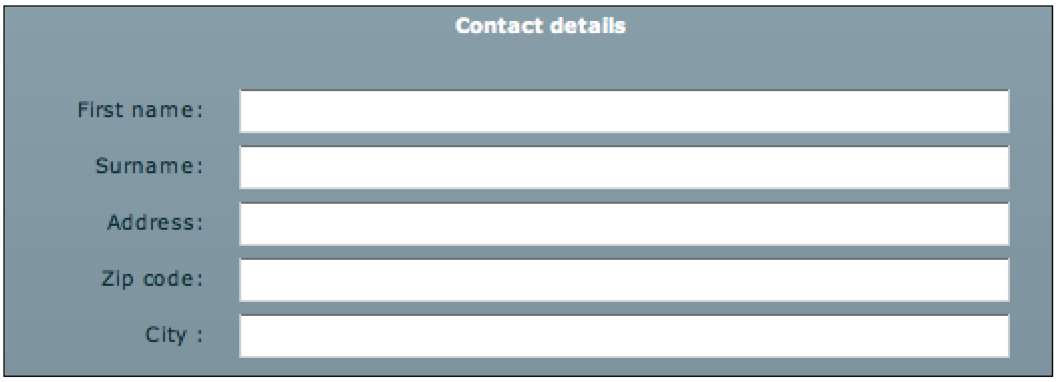
However, 4D for Flex provides more benefits to you using SQLResultSet. Simply Create, Update and Delete records associated with an SQLResultSet, then call `SQLResultSet.submitChanges()`, if needed, to send the changes to the Server. The Server will apply the changes correctly. Only the modified data are sent over the network.

Warning

In the current version (11.2) of 4D SQL Server, if the ResultSet is built from a SELECT joining at least two tables, it will always be non updatable. In this case, you can't use the CRUD functionalities. This limitation will be removed in future versions of 4D SQL Server.

CREATING A RECORD

Let's look at an example in which we can edit a contact in a form:



If we want to add a new row from this form to the current SQLResultSet, we first need to create a Contact object and fill it with the value from the text fields:

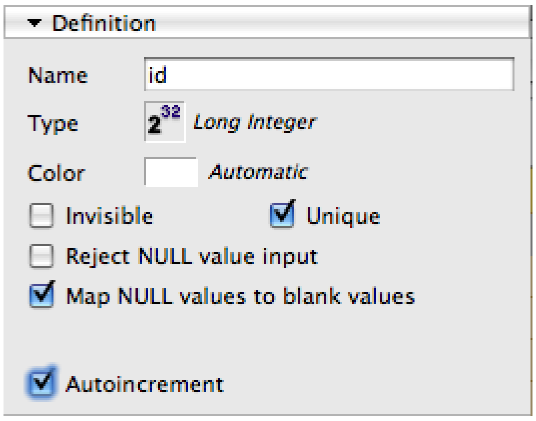
```
var contactObject:Object=new Object();;
contactObject[_resultSet.fields[1].name]=this.firstNameField.text;
contactObject[_resultSet.fields[0].name]=this.surnameField.text;
contactObject[_resultSet.fields[2].name]=this.addressField.text;
contactObject[_resultSet.fields[3].name]=this.zipCodeField.text;
contactObject[_resultSet.fields[4].name]=this.cityField.text;
```

Here we use the fields object property already seen in the "Using SQLResultSet" paragraph.

Then, we will simply call SQLResultSet.insertRecord() to add a new record to the SQLResultSet:

```
        _resultSet.insertRecord(contactObject);
```

Suppose the CONTACTS table in the 4D database has a primary key named ID. Maybe, you don't use the new autoincrement property for a 4D field.



Definition

Name: id

Type: Long Integer

Color: Automatic

[ ] Invisible [✓] Unique

[ ] Reject NULL value input

[✓] Map NULL values to blank values

[✓] Autoincrement

Then, the best way of correctly assigning the ID field is to use a trigger in 4D with the following code:

```
        Case of
            : (Database event=On Saving New Record Event )
                [Contact]id:=Sequence number([Contact])
        End case
```

But keep in mind that you won't get this ID on the Flex side unless you send a new query to the database.

UPDATING A RECORD

On the same basis, we can update some fields of the record currently selected in the dataGrid contact_dg listing the contacts:

```
_resultSet.updateRecordAt(contacts_dg.selectedIndex, 2, Number(total));
```

DELETING A RECORD

Also very easy to do. Suppose we want to delete the current row of the dataGrid. We simply write the following:

```
        _resultSet.deleteRecordAt(contacts_dg.selectedIndex);
```

CONCURRENT ACCESS

When submitting changes or accessing records in write mode by SQL queries, conflicts may occur. If any record that 4D for Flex tries to modify is locked, the whole transaction is cancelled.

USING ASYNCTOKEN

MULTIPLE COMPONENTS

You may use several SQLService components in the same application although it is recommended to only use one component per SQL Server. In other words, you will use a different SQLService if you need to connect to different SQL Servers. If you need to send multiple queries to the same SQL Server, it would be better to use the AsyncToken mechanism.

IT'S ABOUT DIFFERENTIATING THE ORIGIN OF A RESULT

Suppose you need to send several queries to the same database: let's say first for querying a list of companies and filling a combo-box; then when a company is selected in the combo-box to query for the contacts of the selected company. How do you proceed since you only have one event listener for getting the result?

You will use the AsyncToken mechanism in order to be able to differentiate the incoming result.

ASYNCTOKEN CLASS

This solution is not specific to 4D for Flex, it works just the same for other Flex services. See the Adobe documentation:

http://www.adobe.com/livedocs/flex/201/langref/mx/rpc/AsyncToken.html

Package mx.rpc

Class public dynamic class AsyncToken

Inheritance AsyncToken Inheritance EventDispatcher Inheritance Object

This class provides a place to set additional or token-level data for asynchronous RPC operations. The AsyncToken can be referenced in ResultEvent and FaultEvent from the token property.

EXAMPLE

Let's see how we will accomplish that.

First, let's import the corresponding Class.

```
import mx.rpc.AsyncToken;
```

Then, let's declare the AsyncToken:

```
var token:AsyncToken ;
```

And assign it as a result of the SQLService.execute() function:

```
token=fourDSQLService.execute(sql);
token.type='societyCall';
```

Once the token has been instantiated, you can add whatever property you want to it. This property will be kept as part of the token during all of the exchange with the Server.

Don't be afraid of setting the property after calling SQLService.execute(). Asynchronicity is such that it will work perfectly.

When the result event is triggered, the eventListener can get the token and determine the origin of the query:

```
private function resultHandler(event:ResultEvent):void
{
        if(event.token.type=='societyCall')
        {
                // do whatever you want here
        }
}
```

USING SELECT FOR UPDATE

We will use the AsyncToken technique to perform a Select for Update on several records.

First, let's start a transaction:

```
var token:AsyncToken = fourDSQLService.execute("start
transaction");
token.type = "startTransaction";
```

In the result handler, we test the token to perform the following code:

```
if(event.token.type == "startTransaction")
{
        token = fourDSQLService.execute("SELECT id, firstName,
        surname, address, zipCode, city FROM Contact FOR UPDATE");
        token.type = "selectForUpdate";
}
```

Note that:

- We specify FOR UPDATE at the end of the SELECT statement. Doing so, all the records selected will also be locked for other users until we commit or rollback the transaction;
- We change the type of the token to "selectForUpdate."

Now, if the result handler is called with a token whose type is selectForUpdate, we perform the Update statement:

```
if(event.token.type == "selectForUpdate")
{
        token = fourDSQLService.execute("UPDATE Contact set
surname='for update' where id=1");
        token.type = "update";
}
```

Next, we handle the execution of the Update statement and finally can commit the transaction:

```
if(event.token.type == "update")
{
        token = fourDSQLService.execute("commit transaction");
        token.type = "commit";
}
```

DEBUGGING

DEBUGGING USING SQL LOG

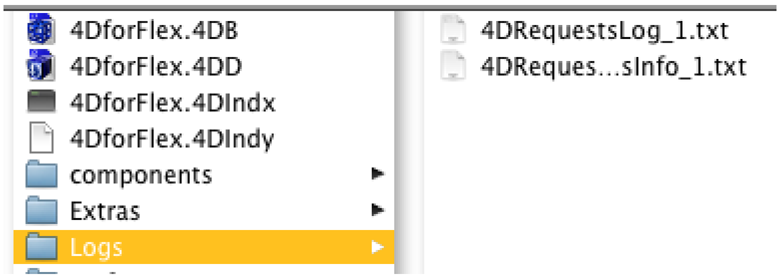
To see what is exchanged between Flex and 4D, it is possible to activate the recording of a log of all SQL requests received by 4D SQL Server.

CLIENT LOG RECORDING CONSTANT

You must use the **SET DATABASE PARAMETER** command with the **<u>Client Log Recording</u>** constant:

- **SET DATABASE PARAMETER(<u>Client Log Recording</u> ;1)** `Starts recording SQL logs on disk
- **SET DATABASE PARAMETER(<u>Client Log Recording</u> ;0)** ` Stops recording SQL logs on disk

The log file is stored in the "Logs" folder near the database structure:



See the 4D manual for more information.

4D CODE EXAMPLE

```
C_BOOLEAN(◊IS_CLIENT_LOG_RECORDING)

If (◊IS_CLIENT_LOG_RECORDING=False)

        SET DATABASE PARAMETER(Client Log Recording ;1)  `starts recording SQL logs on disk

        ◊IS_CLIENT_LOG_RECORDING:=True

Else

        SET DATABASE PARAMETER(Client Log Recording ;0)  `stops recording SQL logs on disk

        ◊IS_CLIENT_LOG_RECORDING:=False

End if
```

LOG EXAMPLE

NORMAL SESSION

Log Session Identifier	B78EDA8185A44DCE9899F5FDDA9F2003

1 Xth log opened on Friday, March 28, 2008 07:23:18 AM

time	task id	component	process info index	request	bytes in	bytes out	duration
03/28/08, 07:23:36	6	'SQLS'	-1		0		
03/28/08, 07:23:36	6	'SQLS'	-1	SQL protocol:LOGIN - OK	0		
03/28/08, 07:23:38	6	'SQLS'	-1	SQL Server execute:SELECT id, firstName, surname, address, zipCode, city FROM Contact	63		
03/28/08, 07:23:38	6	'SQLS'	-1	SQL protocol:EXECUTE_STATEMENT - OK	0		
03/28/08, 07:23:43	6	'SQLS'	-1	SQL protocol:LOGOUT - OK	0		
03/28/08, 07:23:43	6	'SQLS'	-1	SQL protocol:QUIT - OK	0		

INCORRECT SESSION

Log Session Identifier	61D6842F70E142C1BAA09C5E7AA679F2

1 Xth log opened on Wednesday, March 26, 2008 10:09:29 AM

time	task id	component	process info index	request	bytes in	bytes out	duration
03/26/08, 10:09:46	7	'SQLS'	-1		0		
03/26/08, 10:09:46	7	'SQLS'	-1	SQL protocol:LOGIN - OK	0		
03/26/08, 10:09:46	7	'SQLS'	-1	SQL protocol:EXECUTE_STATEMENT - Error:3003 Component:1397836883	0		
03/26/08, 10:09:49	7	'SQLS'	-1		0		
03/26/08, 10:09:49	7	'SQLS'	-1	SQL protocol:LOGIN - OK	0		
03/26/08, 10:09:50	7	'SQLS'	-1	SQL protocol:EXECUTE_STATEMENT - Error:3003 Component:1397836883	0		

SQL PROTOCOL ERRORS

In the 4D manual, you will find an explanation of the error code that may be returned by the Server. For instance in the log above, you see Error: 3003. You can find the meaning of this code in the table below: "Not logged in".

3000	HEADER NOT FOUND

3001	UNKNOWN COMMAND
3002	ALREADY LOGGED IN
3003	NOT LOGGED IN
3004	UNKNOWN OUTPUT MODE
3005	INVALID STATEMENT ID
3006	UNKNOWN DATA TYPE
3007	STILL LOGGED IN
3008	SOCKET READ ERROR
3009	SOCKET WRITE ERROR

CLASS TRACER AND TRACECONSOLE AIR APP

WHEN A TRACE IS NEEDED

Even if you can debug an application directly from FlexBuilder or using a FlashPlayer debug version, you sometimes lack a way of sending messages to a console in a deployment context.

And these times are usually bad times where something goes wrong and you need to understand how your code is working. That's why 4D for Flex helps you by providing:

- A TraceConsole executing as an AIR application;
- An interface to implement if you want to write your own target console;
- Tracer, a static class to send your messages from your Flex code to a console.

TRACER CLASS

Tracer is a static Class. You use it to send messages to a console.

You can trace messages using different methods depending on the nature and severity of the information or error sent to the console:

- Tracer.traceInformation(message:String):void

  Traces information to a console.

- Tracer.traceError(message:String):void

  Traces an error to a console

- Tracer.traceWarning(message:String):void

  Traces a warning to a console

Example:

```
var error:String= "Error #" + event.fault.faultCode;


// Use the Tracer

Tracer.traceError("faultHandler : "+error+" "+
event.fault.faultString);
```

DEFAULT VERBOSE MODE

The default mode of SQLService Class is verbose: it calls the Tracer Class to log events. If no TraceConsole is listening, the messages are lost. So if you want to trace the behavior of your code, there is no code to write, simply open the TraceConsole and click on ‘Connect.’

To stop the automatic tracing, just call Tracer.stop(). Then, Tracer.start() will switch to verbose mode again.

SETTING THE SCOPE OF THE TRACE

When calling the setScope function, you can specify two parameters to describe the kind of information you want to get:

- **Tracer.setScope** (level:Number= Tracer.FLEX_LAYER, type:Number= Tracer.ERROR_TYPE)

  Default mode will trace the error messages of the Flex layer.

If you want to trace all that happens on the AS3 layer, you can call:

```
Tracer.setScope (Tracer.AS3_LAYER, Tracer.ALL);
```

To only trace the warnings on the Flex layer:

```
Tracer.setScope (Tracer.FLEX_LAYER, Tracer.WARNING_TYPE);
```

Values for level:

```
Tracer.FLEX_LAYER
Tracer.AS3_LAYER
Tracer.PROTOCOL_LAYER
```

Values for type:

```
Tracer.INFO_TYPE: only traces information messages
Tracer.WARNING_TYPE: traces the warning messages
Tracer.ERROR_TYPE: traces the error messages
Tracer.NONE: traces no messages at all
Tracer.ALL: traces all messages
```

Caution:

- Beware of the default mode which is set to FLEX_LAYER:

  ```
  Tracer.traceInformation("my important info");
  Tracer.traceError("my fatal error", AS3_LAYER);
  ```

  Only the messages of the Flex layer are sent, so you won’t see the “my fatal error” error.

- This filter is also applied to the message you explicitly send.

  Suppose you write:

  ```
  Tracer.setScope(Tracer.FLEX_LAYER, Tracer.ERROR_TYPE);
  ```

```
        Tracer.traceInformation("my important info");
        Tracer.traceError("my fatal error");
```

Only the error trace will be sent to the console.

When you call `Tracer.traceInformation()`, `Tracer.traceWarning()` or `Tracer.traceError`, you can pass a second parameter specifying the scope:

```
    Tracer.traceInformation("my important info")
```

will be considered as a FLEX_LAYER message, while:

```
    Tracer.traceInformation("my info message", Tracer.AS3_LAYER)
```

will be part of the AS3_LAYER.

If you want to trace several layers, you can write:

```
    Tracer.setScope(Tracer.FLEX_LAYER | Tracer.AS3_LAYER)
```

Then:

```
    Tracer.traceInformation("my info");

    Tracer.traceError("my error", AS3_LAYER);
```

Both messages will be sent.

If you want to trace several types of messages, you can write:

```
Tracer.setScope Tracer.FLEX_LAYER | Tracer.AS3_LAYER,
Tracer.INFO_TYPE
| Tracer.ERROR_TYPE)
```

Information and error messages will be sent.

SETTING THE TARGET OF THE TRACE

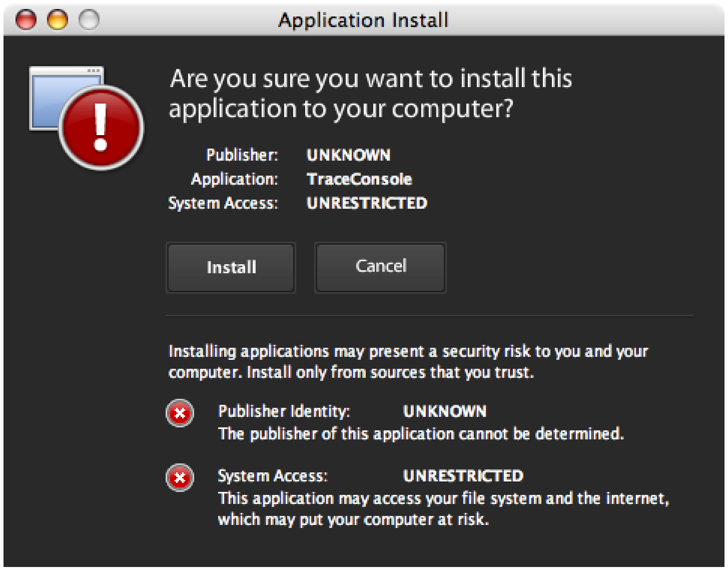
You can specify whether the trace should go to the Flex debugger console while still going to the external console.

Call `Tracer.traceInDebugConsole=true` to display the messages in the Flex debugger console. The default value is False; messages are only sent to the external traceConsole.

USING THE AIR TRACECONSOLE

In order to get the trace messages, a console must be running.

You must install TraceConsole.air.


Application Install

Are you sure you want to install this application to your computer?

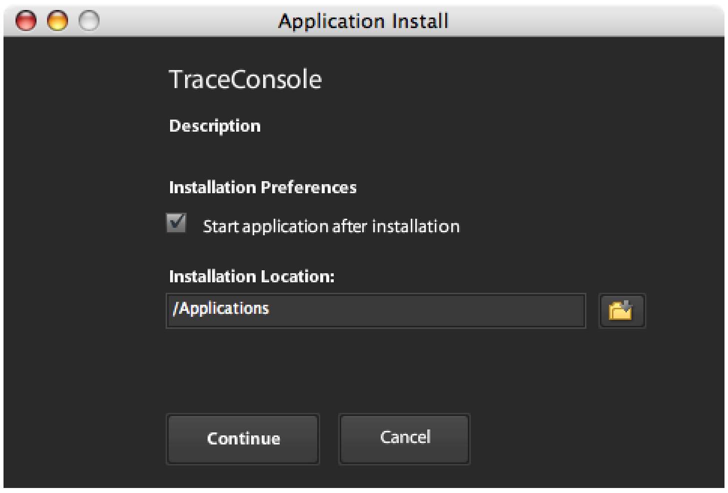
Publisher: UNKNOWN
Application: TraceConsole
System Access: UNRESTRICTED

Installing applications may present a security risk to you and your computer. Install only from sources that you trust.

Publisher Identity: UNKNOWN
The publisher of this application cannot be determined.

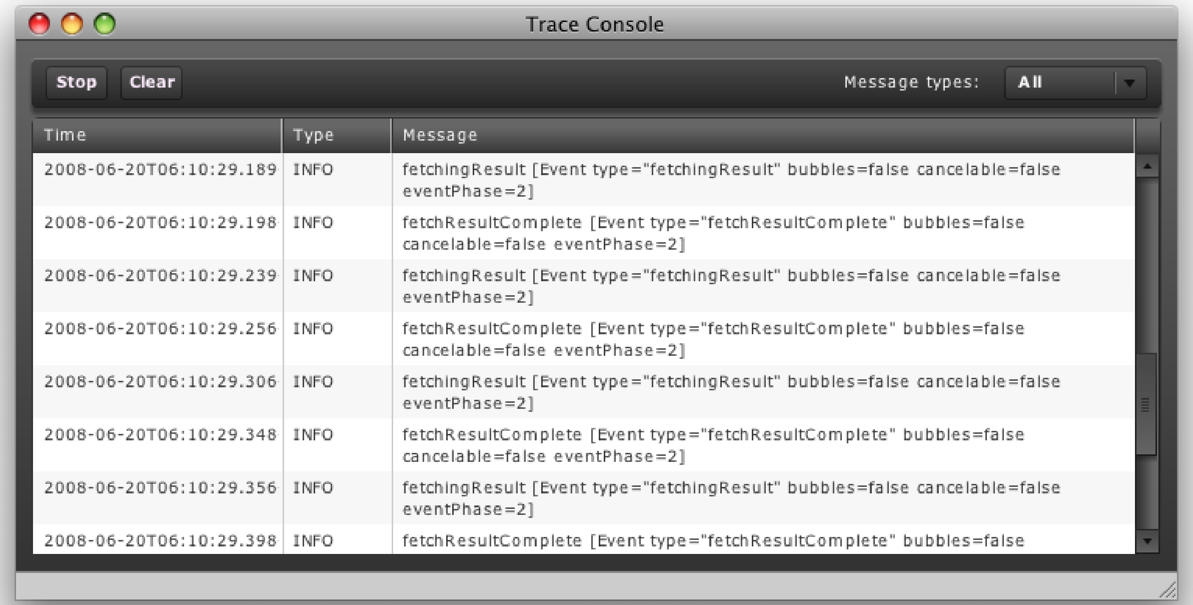
System Access: UNRESTRICTED
This application may access your file system and the internet, which may put your computer at risk.

Click on install, then:



Click on Continue.

This will launch the TraceConsole. Then, click on the 'Listen' button to start listening to incoming messages.



Time	Type	Message
2008-06-20T06:10:29.189	INFO	fetchingResult [Event type="fetchingResult" bubbles=false cancelable=false eventPhase=2]
2008-06-20T06:10:29.198	INFO	fetchResultComplete [Event type="fetchResultComplete" bubbles=false cancelable=false eventPhase=2]
2008-06-20T06:10:29.239	INFO	fetchingResult [Event type="fetchingResult" bubbles=false cancelable=false eventPhase=2]
2008-06-20T06:10:29.256	INFO	fetchResultComplete [Event type="fetchResultComplete" bubbles=false cancelable=false eventPhase=2]
2008-06-20T06:10:29.306	INFO	fetchingResult [Event type="fetchingResult" bubbles=false cancelable=false eventPhase=2]
2008-06-20T06:10:29.348	INFO	fetchResultComplete [Event type="fetchResultComplete" bubbles=false cancelable=false eventPhase=2]
2008-06-20T06:10:29.356	INFO	fetchingResult [Event type="fetchingResult" bubbles=false cancelable=false eventPhase=2]
2008-06-20T06:10:29.398	INFO	fetchResultComplete [Event type="fetchResultComplete" bubbles=false

Tracing execution in the Trace Console

Click on Stop when you don't want to listen to incoming trace messages any longer.

You can't execute several TraceConsoles simultaneously: all messages will be captured by one console.

If you run several Flex/Air apps, all messages are logged in the same and unique TraceConsole.

One line of trace is composed of fields separated by tabs, so you can copy traces and paste them into Excel.

USING TRACER CLASS

Tracer class is part of the Flex4D_SQL package. Simply use the methods of the Tracer Class to send messages to the TraceConsole:

```
Tracer.traceInformation("some text");
```

This will display "some text" in the TraceConsole.

The `Tracer.started` property indicates whether the tracer is activated. If not, simply call `Tracer.start()`.

You can add an EventListener to find out whether the message was sent to the console correctly:

```
addEventListener(type:String, listener:Function, useCapture:Boolean =
false, priority:int = 0.0, useWeakReference:Boolean = false):void
```

If the message is correctly sent, the level property of the status event object is “status”; otherwise, it is “error.”

INTERFACE

If you're not pleased with the TraceConsole AIR app provided, you can write one of your own. You only need to implement the sltraceReceiver interface.

SECURITY

Security of transactions on a public network is a major concern.

4D for Flex allows authentication by sending user-password to the Server. This information can be virtual as, on Server side, the 4D Developer can check the data transmitted in the call-back method **On SQL Authentication**. There, he can deny authorization or change the user used to access to the SQL Server.

SECURITY CONCERNS IN THE FLASH PLAYER

“The Flash Player provides a security sandbox to isolate applications and the host operating system. The Flash Player sandbox security has a significant advantage over traditional web-enabled application components, such as ActiveX, that require users to trust code that often has complete access to the operating system environment. The default configuration of the Flash Player’s sandbox provides security by segmenting execution privileges between the Flex application and the operating system. Similar to Java’s sandbox technology, the Flash Player’s sandbox prevents unauthorized access to the operating system environment as well as other local instances of the Flash player” (see the Adobe Doc).

The Flash Player security policy prevents a Flex application from sending a request to a domain other than the one which previously served the SWF file.

If your SWF was served by mydomain.com you cannot send queries to mydomainSQL.com, for example.

So, if you serve the SWF from an Apache HTTP server on mydomain.com for instance, by default you won't be able to send queries on the 4D SQL Server running on mydomainSQL.com. This issue can be addressed in several ways:

- The containing HTML page may be served by the Apache Server and request an SWF in the mydomainSQL.com domain, using the 4D built-in HTTP Server.
- You can use a cross-domain policy file to specify the origins of the query you want to admit. See the Adobe doc on Security Topic Center : http://www.adobe.com/devnet/security/

SECURITY UPDATE STARTING FROM FLASH PLAYER V 9.0.124

PRIOR VERSIONS

This policy-file allowed any website to access the database from a Flex app.

```
<?xml version='1.0'?>

<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">

<cross-domain-policy>

      <allow-access-from domain='*' to-ports='1919' />

</cross-domain-policy>
```

NEWS SECURITY SCHEMA

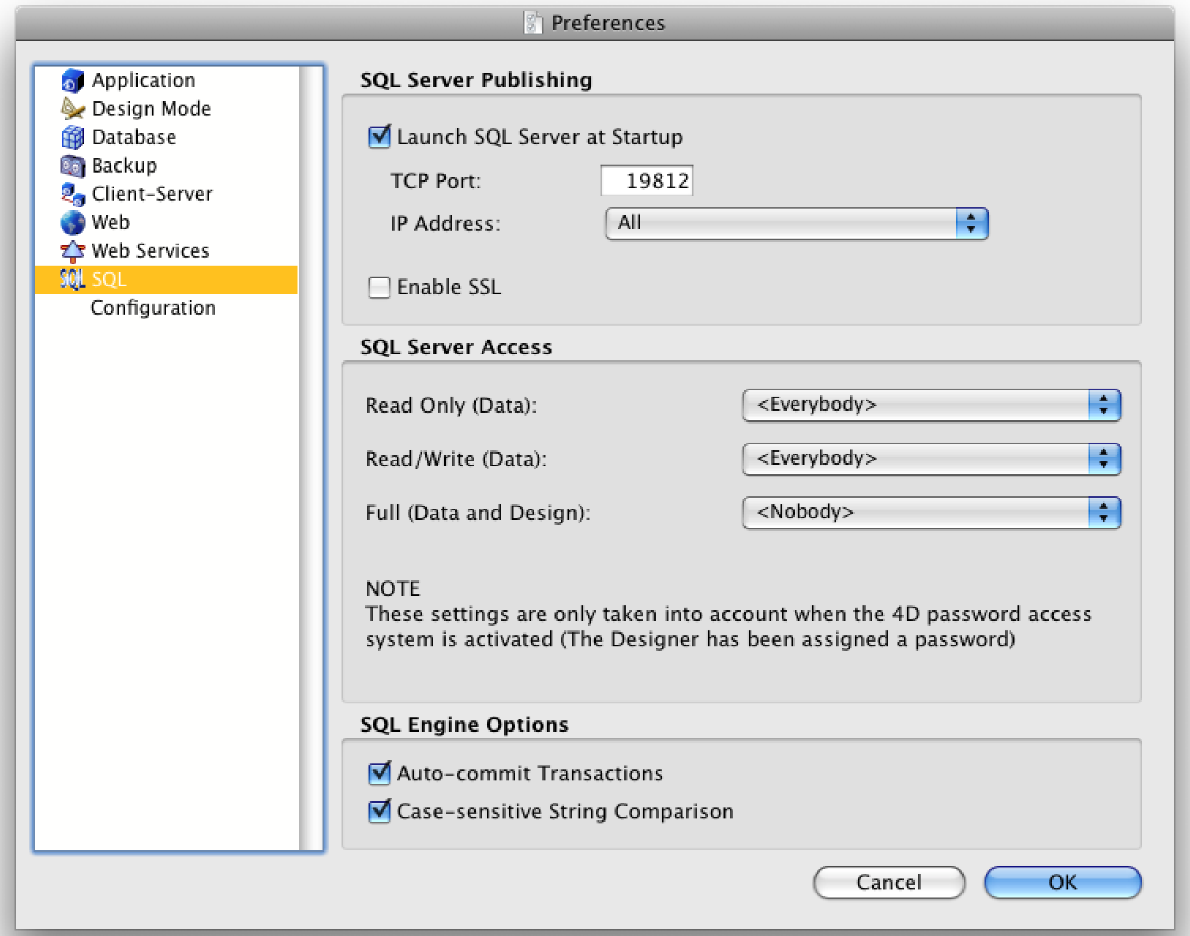
The Flash Player now distinguishes connections using binary sockets from “url connections”. See: http://www.adobe.com/support/security/bulletins/apsb08-11.html

In order to allow a Flash Player to connect to a 4D SQL Server, you must place a specific policy file called 'socketpolicy.xml' at a given place, in a folder inside the DB package: 'Preferences/SQL/Flash'. The content of this file must be similar to this to allow all incoming connections to port 19812:

```
<cross-domain-policy>
 <allow-access-from domain="*" to-ports="19812"/>
</cross-domain-policy>
```

SETTING SQL PREFERENCES ON THE 4D SIDE

Several settings are available for SQL in the 4D preferences.



SQL Server Publishing

[✓] Launch SQL Server at Startup

TCP Port: 19812

IP Address: All

[ ] Enable SSL

SQL Server Access

Read Only (Data): <Everybody>

Read/Write (Data): <Everybody>

Full (Data and Design): <Nobody>

NOTE
These settings are only taken into account when the 4D password access system is activated (The Designer has been assigned a password)

SQL Engine Options

[✓] Auto-commit Transactions

[✓] Case-sensitive String Comparison

The ‘Enable SSL’ check box has no effect on 4D for Flex communications since SSL is not available at socket level in ActionScript.

SQL SERVER ACCESS

It is possible to restrict access rights for SQL queries to certain groups of users. For instance, members of the ‘sqlGroup’ may be allowed to Read/Write Data, but only the members of the ‘Administration’ group will be allowed to use data manipulation language (DML).

See the 4D documentation for more details.

ON SQL AUTHENTICATION

When a 4D for Flex client tries to log in to 4D, the **On SQL Authentication** database method is triggered. In this method, the 4D developer can:

- Set the 4D user that will be used to determine access rights using **CHANGE CURRENT USER**;
- Accept or reject the login.

See the 4D documentation for more details.

ADDENDUM

BEST PRACTICES FOR WORKING WITH SQL DATABASES

Adobe lists in its doc some best practices for working with SQL Databases in AIR: "Developing AIR applications with Flex / Files and data / Working with local SQL databases / Strategies for working with SQL databases".

http://livedocs.adobe.com/flex/3/html/help.html?content=SQL_20.html

Most of them equally apply to communications between Flex and 4D.

DATATYPES

This table compares 4D and AS3 native types.

4D types	AS3 types	Additional information
Alpha	String	identical match
Text	String	identical match
Date	Date	If a Date field is created/updated with the 4D language, the time part is always set to 0 for a SQL query. If a Date field is created/updated through a SQL query, the time part can be set but will be unavailable with the 4D language.
Time	fourD.values.VDuration	If a Time field is created/updated with the 4D language, the millisecond part is always set to 0 for a SQL query. If a Time field is created/updated through a SQL query, the millisecond part can be set but will be unavailable with the 4D language.
Boolean	Boolean	identical match
Integer	int	32-bit integer
Long Integer	int	32-bit integer
Integer 64 bits	Number	64-bit integer in 4D 53-bit integer in AS3 (due to the limited integer range of the Number class)
Real	Real	IEEE-754 double-precision floating-point number
Float	*unsupported*	*unsupported*
Blob	fourD.values.VBlob	identical match
Picture	fourD.values.VImage	identical match