



**PAVING THE WAY**

MIVA SMALL BUSINESS CONFERENCE 2006  
AUG. 29 - SEPT. 1 • SAN DIEGO, CA.

# MIVA Small Business Conference 2006

## **Building MIVA Script Database Applications**

**latu.net**

CREATIVE DEVELOPMENT



# **MIVA Script as a Development Platform**

MIVA Script is a robust development environment for integrating databases into a web site. The MIVA Empresa Virtual Machine has a built-in database server, and it can be used for accessing third-party database servers such as MySQL.

This session will cover methods for accessing databases and developing dynamic data-driven sites.

# MIVA SCRIPTING DEFINED

- MIVA Script is an XML-based server-side scripting language that is implemented as a pre-processor for other Web based languages. MIVA Script provides XML-compliant tags for database, commerce, control flow, modularization, and easy access to protocols
- MIVA Script programs are HTML documents that also contain tags (commands) from the MIVA Script programming language. MIVA Script is a server-side scripting language that is implemented by MIVA Empresa (and MIVA Mia) rather than by the browser.
- The MIVA Scripting code is embedded within the HTML code. That is what is meant by "XML-based server-side scripting language". This is in contrast to other languages, like Perl, where the html code may be embedded within the programming code.

# MIVA SCRIPTING SYNTAX

- Miva Script tags are XML-based: they have the same format as HTML tags; the element names (for example, <MvASSIGN>) indicate their function; and the attributes specify values that the tags operate on.
- Miva Script tags correspond to typical programming language constructs such as assignment statements, conditional expressions, loops, and input/output statements, as well as Miva Script's special database, mail, and commerce functionality.

```
<MvASSIGN NAME = "l.ok" VALUE = "{ 1 }">
```

```
<MvIF Expr = "{ l.this }">
```

```
  ...  
<MvELSE>
```

```
  ...  
</MvIF>
```

# MIVA SCRIPTING TAGS

- **MvASSIGN** - assigns a value to a variable.

```
<MvASSIGN NAME = "l.ok" VALUE = "{ 1 }">
```

- **MvEVAL** - evaluates an expression; result is displayed.

```
<MvEVAL EXPR="{ '<p>' $ l.varname $ '</p>' }">
```

- **Arrays** - values can be signed to arrays.

```
<MvASSIGN NAME = "l.i" VALUE = "{ 1 }">
```

```
<MvASSIGN NAME="l.array_name" INDEX="{ l.i }" VALUE="{ 'why' }">
```

```
<MvEVAL EXPR="{ l.array_name[1] }">
```

- **Structures** - name-value pairs can be assigned to structures.

```
<MvASSIGN NAME="l.struct" MEMBER="code" VALUE="{ 'blue' }">
```

```
<MvASSIGN NAME="l.struct" MEMBER="label" VALUE="{ 'Blue' }">
```

```
<MvEVAL EXPR="{ l.struct:code }">
```

# MIVA SCRIPTING TAGS

- **MvIF, MvELSE** - tests conditions, controls flow of program.

```
<MvIF EXPR = "{ (l.action EQ 'A') OR (l.action EQ 'B') }">  
  do something...  
<MvELSEIF EXPR = "{ l.action EQ 'C' }">  
  do something...  
<MvELSEIF EXPR = "{ l.action EQ 'D' }">  
  do something...  
<MvELSE>  
  or do this...  
</MvIF>
```

- **MvWHILE** - looping flow control for repeatedly processing a section of code.

```
<MvASSIGN NAME = "l.i" VALUE = "{ 1 }">
```

```
<MvWHILE EXPR = "{ l.i LT l.foo }">
```

do something a certain number of times, depending on expression

```
<MvASSIGN NAME = "l.i" VALUE = "{ l.i + 1 }">  
</MvWHILE>
```

# MIVA SCRIPTING TAGS

- **MvFUNCTION**

- Useful for separating code into components, creating a top-down modularized script.
- Create a layer of abstraction and help make programming "problems" easier to solve.
- Self documenting, making scripts easier to read

```
<MvASSIGN NAME = "l.result" VALUE = "{ FunctionName('one', 'two') }">
```

```
<MvFUNCTION NAME = "FunctionName"  
  PARAMETERS = "blue, purple"  
  STANDARDOUTPUTLEVEL = "text, html, compresswhitespace"  
  ERROROUTPUTLEVEL = "syntax, expression, runtime">
```

```
<MvCOMMENT>
```

Notice that the parameters are declared without prefixes.

They then have the local prefix when they are used within the function.

```
</MvCOMMENT>
```

```
<MvASSIGN NAME = "l.output" VALUE = "{ l.blue + l.purple }">
```

```
<MvFUNCTIONRETURN VALUE = "{ l.output }">
```

```
</MvFUNCTION>
```

# MIVA CLASSIC DATABASE

- The old school method for accessing xbase databases.
- Operation specific database tags:
  - <MvCREATE>
  - <MvOPEN>
  - <MvCLOSE>
  - <MvADD>
  - <MvUPDATE>
  - <MvDELETE>
  - <MvUNDELETE>
  - <MvPACK>
  - <MvPRIMARY>
  - <MvMAKEINDEX>
  - <MvSETINDEX>
  - <MvFIND>
  - <MvFILTER>
  - <MvSKIP>
  - <MvREVEALSTRUCTURE>

# MIVA SQL DATABASE

- Miva SQL (mivasql)
  - internal
  - xbase data files
- MySQL (mysql)
  - external
  - remote
  - database server
- Generalized database tags with SQL queries:
  - <MvOPEN>
  - <MvQUERY>
  - <MvOPENVIEW>
  - <MvREVEALSTRUCTURE>
- A much more flexible approach.

# DATABASE TERMINOLOGY

- **Field** - the structure of a database table is composed of fields; each field has a name & type; some types also have a size;

Examples:

id	NUMBER(8),
name	CHAR(60),
descrip	MEMO,
active	BOOL,
date	DATE

- **Record** - a set of values; also called a row; each record contains a value for each field in the database table;
- **Index** - a method for sorting or re-ordering a database table; a database table can be indexed by any of its fields;
- **Alias** or **View** - a name that is used for referring to a database table; this name can be different than the name of the database table; a database can be referred to by one or more aliases;

# CREATING A DATABASE TABLE

```
<MvQUERY
  NAME = "Merchant"
  QUERY = "{ 'CREATE TABLE Cats' '$ ' '$ (' '$
'id      '$ [ g.Native_DBAPI ].DB_Type_NUMBER( 0,0 )    '$ ',' '$
'code   '$ [ g.Native_DBAPI ].DB_Type_CHAR( 50 )        '$ ',' '$
'name   '$ [ g.Native_DBAPI ].DB_Type_CHAR( 100 )       '$ ',' '$
'type   '$ [ g.Native_DBAPI ].DB_Type_CHAR( 8 )         '$ ',' '$
'descrip '$ [ g.Native_DBAPI ].DB_Type_MEMO()           '$ ',' '$
'order  '$ [ g.Native_DBAPI ].DB_Type_NUMBER( 0,0 )     '$ " '$
'date   '$ [ g.Native_DBAPI ].DB_Type_CHAR( 10 )        '$ " '$
')' }">
```

- The DB\_Type\_ parts of the query are actually function calls to a database library.
- g.Native\_DBAPI is the location of the Database API library.
  - lib/dbapi\_mysql.mvc is the library for Miva Merchant
- DB\_Type\_CHAR is a function within the library.

# NOTES ON CREATING A TABLE

- When creating databases in a stand-alone script, the `g.Native_DBAPI` variable will need to be assigned a value.
- When creating databases in a Miva Merchant module use the `g.Module_Library_DBAPI` variable.

# OPENING A DATABASE TABLE

## Description

- <MvOPEN>
- <MvQUERY>
- The database is opened once.
- Then each database table is queried.

## Example

```
<MvOPEN NAME      = "Merchant"  
      DATABASE    = "{ g.DB:Database }"  
      USERNAME    = "{ g.DB:Username }"  
      PASSWORD    = "{ g.DB:Password }"  
      TYPE        = "{ g.DB:Type }">
```

```
<MvOPENVIEW NAME  = "Merchant"  
      VIEW        = "Cats"  
      QUERY      = "{ 'SELECT * FROM Cats ' $  
                    'ORDER BY id ' $  
                    'LIMIT 0 , 10' }">
```

Example: [SQL Open Database Script](#)

# FINDING A RECORD

- **MvOPENVIEW SELECT**

```
<MvOPENVIEW  
  NAME           = "Merchant"  
  VIEW           = "Cats"  
  QUERY         = "{ 'SELECT * FROM Cats WHERE id = ?' }"  
  FIELDS        = "l.id">
```

# FINDING A RECORD, Example

```
<MvFUNCTION NAME = "Cats_Load_ID" PARAMETERS = "id, cat var" STANDARDOUTPUTLEVEL = "">
  <MvASSIGN NAME = "l.ok" VALUE = "{ 1 }">
<MvIF Expr = "{ l.id }">
  <MvOPENVIEW NAME = "Merchant"
    VIEW = "Cats"
    QUERY = "{ 'SELECT * FROM ' $ g.Store_Table_Prefix $ 'Cats WHERE id = ?' }"
    FIELDS = "l.id">
  <MvIF Expr = "{ g.MvOPENVIEW_Error }">
    <MvASSIGN NAME = "l.ok" VALUE = "{ 0 }">
  <MvELSE>
    <MvIF Expr = "{ Cats.d.EOF }">
      <MvASSIGN NAME = "l.ok" VALUE = "{ 0 }">
    <MvELSE>
      <MvASSIGN NAME = "l.cat:id" VALUE = "{ Cats.d.id }">
      <MvASSIGN NAME = "l.cat:code" VALUE = "{ Cats.d.code }">
      <MvASSIGN NAME = "l.cat:name" VALUE = "{ Cats.d.name }">
    </MvIF>
  </MvIF>
  <MvCLOSEVIEW NAME = "Merchant" VIEW = "Cats">
<MvELSE>
  <MvASSIGN NAME = "l.ok" VALUE = "{ 0 }">
</MvIF>
<MvFUNCTIONRETURN VALUE = "{ l.ok }">
</MvFUNCTION>
```

# INSERTING A RECORD

- **MvQUERY INSERT**

```
<MvQUERY
  NAME          = "Merchant"
  QUERY         = "{ 'INSERT INTO Cats
                    ( id, code, name, descip, active, lastupdate )
                    VALUES
                    ( ?, ?, ?, ?, ?, ? )' }"
  FIELDS       = "l.data:id,
                  l.data:code,
                  l.data:name,
                  l.data:descip,
                  l.data:active,
                  l.time_t">
```

# INSERTING A RECORD, Example

```
<MvFUNCTION NAME = "Cats_Insert" PARAMETERS = "cat var" STANDARDOUTPUTLEVEL = "">
```

```
  <MvASSIGN NAME = "l.ok" VALUE = "{ 1 }">
```

```
  <MvQUERY NAME = "Merchant"
```

```
    QUERY = "{ 'INSERT INTO ' $ g.Store_Table_Prefix $ 'Cats
```

```
      ( id, code, name )
```

```
      VALUES
```

```
      ( ?, ?, ? )' }"
```

```
    FIELDS = "l.cat:id,
```

```
      l.cat:code,
```

```
      l.cat:name">
```

```
  <MvIF Expr = "{ g.MvQUERY_Error }">
```

```
    <MvIFDEF NAME="DEBUG">
```

```
      <MvEVAL Expr="{ 'g.MvQUERY_Error: ' $ g.MvQUERY_Error $ '<br>' }">
```

```
    </MvIFDEF>
```

```
    <MvASSIGN NAME = "l.ok" VALUE = "{ 0 }">
```

```
  </MvIF>
```

```
  <MvFUNCTIONRETURN VALUE = "{ l.ok }">
```

```
</MvFUNCTION>
```

# UPDATING A RECORD

- **MvQUERY UPDATE**

```
<MvQUERY
  NAME          = "Merchant"
  QUERY         = "{ 'UPDATE Cats
                    SET
                      code      = ?,
                      name      = ?,
                      descip    = ?,
                      active    = ?,
                      lastupdate = ?
                    WHERE
                      id        = ?' }"
  FIELDS        = "l.data1:code,
                  l.data1:name,
                  l.data1:descip,
                  l.data1:active,
                  l.time_t,
                  l.data1:id">
```

# UPDATING A RECORD, Example

```
<MvFUNCTION NAME = "Cats_Update" PARAMETERS = "cat var" STANDARDOUTPUTLEVEL = "">
```

```
<MvASSIGN NAME = "l.ok" VALUE = "{ 1 }">
```

```
<MvIF EXPR = "{ l.cat.id }">
```

```
<MvQUERY NAME = "Merchant"
```

```
  QUERY = "{ 'UPDATE ' $ g.Store_Table_Prefix $ 'Cats
```

```
  SET
```

```
    code = ?,
```

```
    name = ?
```

```
  WHERE
```

```
    id = ? }"
```

```
  FIELDS = "l.cat:code,
```

```
  l.cat:name">
```

```
<MvIF EXPR = "{ g.MvQUERY_Error }">
```

```
<MvIFDEF NAME="DEBUG">
```

```
<MvEVAL EXPR="{ 'g.MvQUERY_Error: ' $ g.MvQUERY_Error $ '<br>' }">
```

```
</MvIFDEF>
```

```
<MvASSIGN NAME = "l.ok" VALUE = "{ 0 }">
```

```
</MvIF>
```

```
<MvELSE>
```

```
<MvASSIGN NAME = "l.ok" VALUE = "{ 0 }">
```

```
</MvIF>
```

```
<MvFUNCTIONRETURN VALUE = "{ l.ok }">
```

```
</MvFUNCTION>
```

# DELETING A RECORD

- **MvQUERY DELETE**

```
<MvQUERY  
  NAME           = "Merchant"  
  QUERY          = "{ 'DELETE FROM Cats WHERE id = ?' }"  
  FIELDS        = "l.id">
```