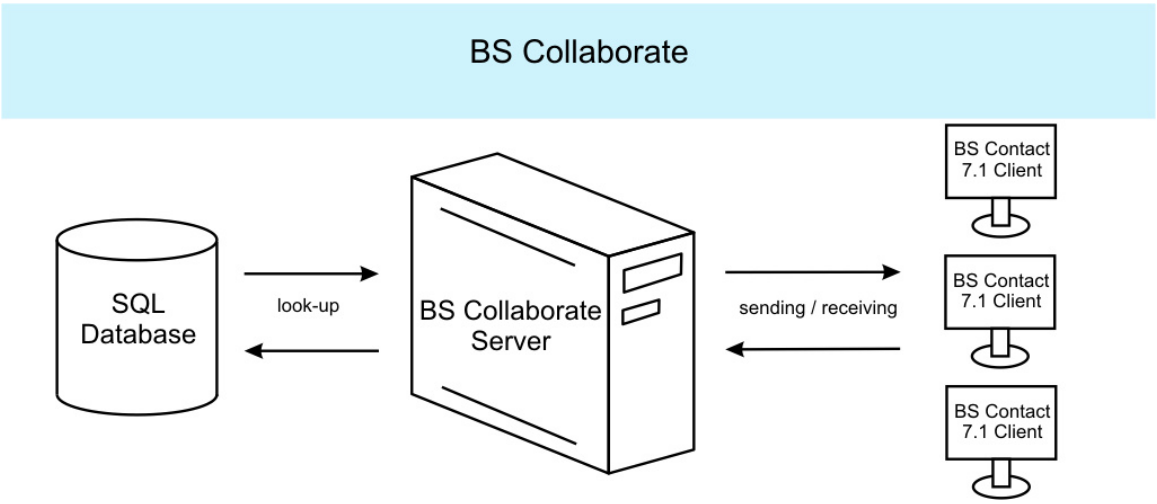
Bitmanagement Software GmbH

BS Collaborate



Disclaimer

THIS SOFTWARE DISCRIPTION IS PROVIDED BY BITMANAGEMENT SOFTWARE GMBH "AS IS" WITHOUT WARRANTY OF ANY KIND AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE DISCLAIMED. NO WARRANTY FOR ANY PROBLEM ARISING OUT OF A DOWNLOAD AND/OR USE OF THIS SOFTWARE CAN BE UNDERTAKEN. SUBJECT TO REGULATORY AND ANY OTHER OBLIGATIONS AND LIABILITIES WHICH ARE NOT PERMITTED TO BE EXCLUDED, UNDER NO CIRCUMSTANCES SHALL BITMANAGEMENT SOFTWARE GMBH OR AFFILIATED PARTIES BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY SPECIAL, PUNITIVE, INCIDENTAIL INDIRECT OR CONSEQUENTIAL DAMAGE. THE DOWNLOAD AND/OR USE OF THIS SOFTWARE IS RESTRICTED TO NONCOMMERCIAL USE AND AN EVALUATION PERIOD. ANY ADDITIONAL FEE BY THIRD PARTIES THAT MIGHT OCCUR FOR DOWNLOAD OR USAGE OF THE SOFTWARE HAS TO BE BORN BY THE PARTY DOWNLOADING OR USING THE SOFTWARE.

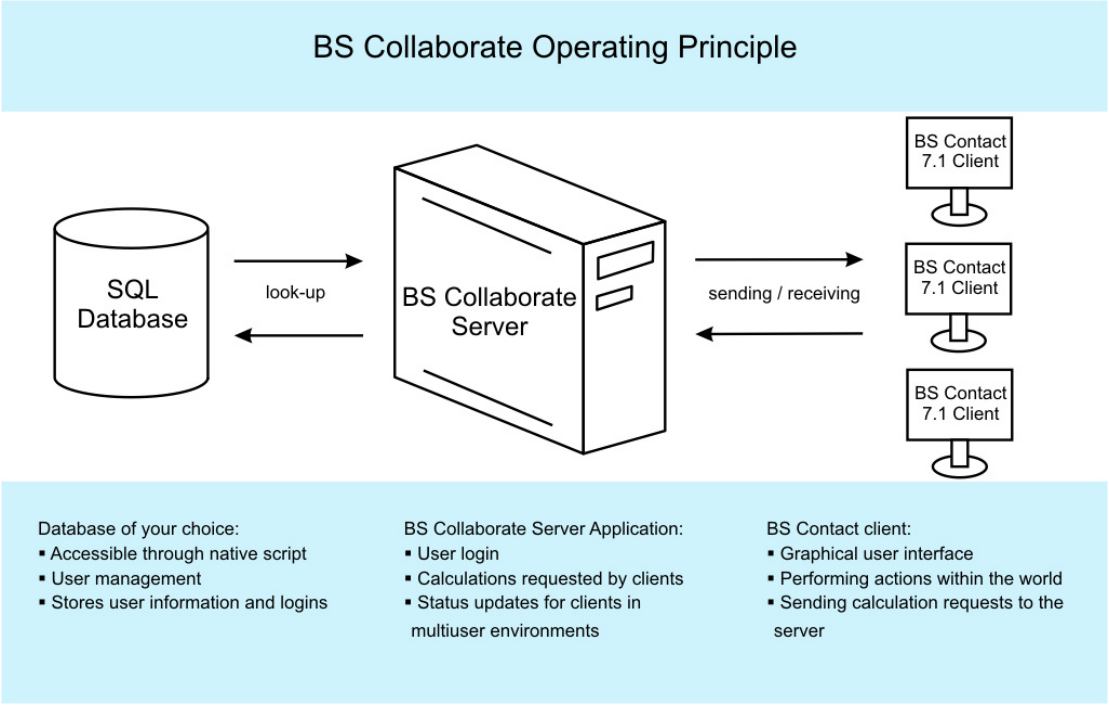
Bitmanagement Software GmbH
Oberlandstraße 26
D-82335 Berg
Germany
Phone: 0049(0)8151-971708
Fax: 0049(0)8151-971709
E-mail: info@bitmanagement.de
Web: www.bitmanagement.de

Bitmanagement Software GmbH

Table of Contents

This description is focusing on advanced person with basic knowledge in VRML/X3D/Collada. It is also advantageous to have access to the BS SDK.

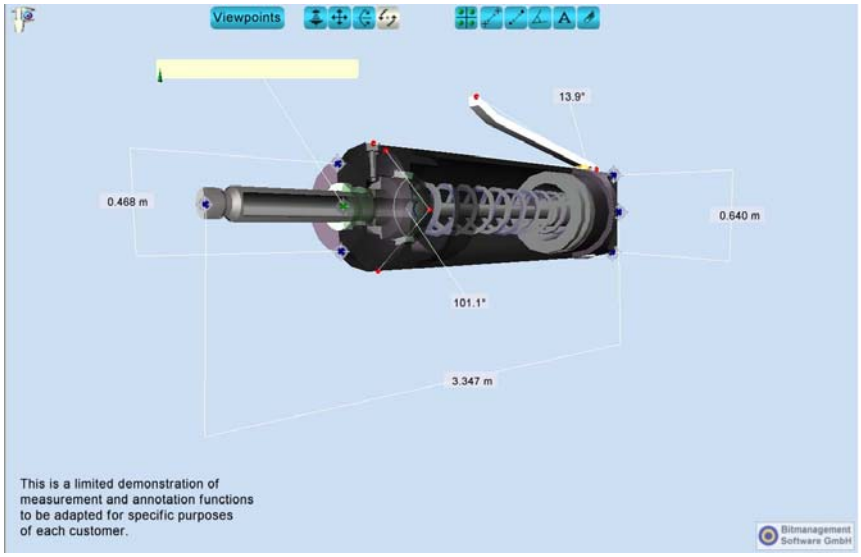
1 Collaborate server - client structure



The Bitmanagement Software collaborate System is build up on the Web3D proposal for the Networking component (see networkSensor.html). These nodes allow VRML/X3D scenes to connect to arbitrary servers or direct links between two VRML/X3D players. With these nodes you are able to manipulate virtual objects collaboratively in real time. The Bitmanagement Software node extension allows you to communicate during these manipulations. For permit access only registered users you are able to connect your user database to the BS Collaborate server. With the internal login interface there is no hassle of external user management with websites or something comparable. These multi user features are included in BS Contact (VRML/X3D/Collada) 7.1. This means that on client side only the BS Contact 7.1 must be installed to connect with a multi user environment.

2 Supported Products

Professional CAD Systems are able to export 3D models to VRML/X3D. With the BS Collaborate Software you can discuss about the object or measure, describe a construct virtual object interactive and collaboratively in 3D. You can handle the communication over the supplied text chat or e.g. per third party voice chat.



Example for a collaborative measurement

With the BS Collaborate System you are also able to create your own multi user environment with entertainment content. Make advertises or presentations over the internet without being on the same place.

Example for a multi user soccer game

Supported Features in this version:

– Platform independent server

– Shared Events

– Support customized Login screen in 3D

– Text Chat included

– Support customized Chat display in 3D

– Simple server side computation

– Each client has its own Session number for client identification.

– Each event is storable in database or file system

– Compatible with Web3D network proposal

Many more features under development.

3 BS Collaborate Startup

3.1 Authoring a 3D scene for BS Collaborate

On the scene authoring side, the BS Collaborate server implementation is based on a node that specifies the connection parameters for the server connection and a node that handles session based messages to/from the server. These messages inform the scene when a user has joined, when other users move, about chat messages, etc. The scene can then respond to these messages, e.g. by showing a buddy list.

An example scene may look like this:

```
DEF MU BSCollaborate
{
   connection NetConnection
   {
      address "test.bitmanagement.de"
      port 12345
   }
}

ROUTE SomeScript.credentials TO MU.tryLogin
ROUTE MU.loginResult TO SomeScript.loggedIn
...
```

After initialization the BSCollaborate node uses the associated Connection node to connect to the server. It than waits untill reception of the user name and password on the tryLogin field. When it receives those, it uses them to authenticate with the server and establish an identity. This way a 3D scene can show a login screen asking for login and password directly in the 3D window. No external HTML form or similar is required, however will probably later be supported. Alternatively a Script node could just send some constant values to BSCollaborate.tryLogin if no user identification is required.

After a user has logged in, the BSCollaborate node populates its users field with nodes describing all other users in currently logged in. For each user in the scene an event is sent over the hasJoined starting with the own user. The BSCollaborate node in other clients will send add a record to their users field and send an event over their hasJoined field, so that all clients are informed of our arrival.

Similarly, if later during a session a new user joins or leaves the session, this is indicated via hasJoined and hasLeft, and the users is updated by adding a new node or removing the one corresponding to a leaving user. This way a 3D scene can freely query the list of currently connected users and respond to changes in that list, and implementing things like a buddy list.

In the same manner chat messages that the scene collects from an input line in 3D - maybe displayed on a HUD - can be sent to the meSay field and will then be distributed to other clients. If another users utters some chat message, this will be indicated via the hasSaid field. The 3D scene can then display chat messages, e.g. on a HUD, as balloon above avatars, or in whatever way they like.

The node that describes a user participating in the session looks as follows:

```
UserData
{
   field SFInt32     idx      -1
   field SFString    nickname ""
   field SFString    avatarString ""

   eventOut SFString loginState

   eventOut SFVec3f     pos
   eventOut SFRotation  ori
   eventOut SFBool      isMoving

   eventOut MFString    chat

   field    SFNode      userData NULL
}
```

The users field of the BSCollaborate node is filled with nodes of that signature, and the hasJoined, hasLeft, hasMoved and hasSaid, which are of type SFNode emit these nodes. A scene can either respond to the events of the BSCollaborate node or build individual routes directly from the eventOut fields of the UserData.

3.2 Sharing Events between Clients / Maintainung Scene State

The EventStreamSensor node is used to synchronize scene states with other clients. Such a scene state may be the state of a door that can be open or closed, or the position of the ball in a soccer game. Instead of calculating e.g. the ball position by a Script node and sending it to a Transform node via a ROUTE statement, the event is sent to an eventIn field of the EventStreamSensor node. The EventStreamSensor node sends the event to the BS Collaborate server, which then sends it to all other clients, including the originating one. Then the EventStreamSensor in all clients sends the event on an eventOut field to the proper Transform node.Similar to a Script node the EventStreamSensor allows to add arbitrary fields, e.g:

```
DEF Streamer EventStreamSensor
{
   eventIn  SFVec3f  set_BallPos
   eventOut SFVec3f     BallPos_changed
   eventIn  SFBool   set_DoorState
```

```
   eventOut SFBool      DoorState_changed
}
```

Similar to a Script node the EventStreamSensor allows to add arbitrary fields. EventIns and eventOuts are associated if they have the same base name. If the scene sends a value to a set_* eventIn field in one client, the corresponding *_changed eventOut field emits this value in all clients.

The following is a scene without multi-user capabilities:

```
DEF Calculator Script
{
   eventOut SFVec3f BallPosition

   url "vrmlscript: ... "
}

DEF TrBall Transform
{
   children Shape { ... geometry of the ball }
}

ROUTE Calculator.BallPosition TO TrBall.Transform
```

It consists of a Script node, a Transform node and a ROUTE transporting the calculations from the Script to the Transform. For making this multi-user one will add an EventStreamSensor and will break up the ROUTE into two ROUTEs that go through the EventStreamSensor node:

```
DEF Calculator Script
{
   eventOut SFVec3f BallPosition

   url "vrmlscript: ... "
}

DEF TrBall Transform
{
   children Shape { ... geometry of the ball }
}


DEF Streamer EventStreamSensor
{
   connection USE Conn # the same Connection node as used in the BSCollaborate node.

   eventIn  SFVec3f  set_BallPos
   eventOut SFVec3f      BallPos_changed

   ... other fields if necessary
}


# ROUTE Calculator.BallPosition TO TrBall.translation
ROUTE Calculator.BallPosition TO Streamer.set_BallPosition
ROUTE Streamer.BallPosition_changed to TrBall.translation
```

This way if the Calculator node in one client calculates a new ball position, it will be distributed to all TrBall Transform nodes in all clients.

3.3 Distributed Environment

Care must be taken because the system is now a distributed environment. Although the scene contains only one Calculator node, there is an instance of it in every client, and each of them may calculate a position animation for the ball at the same time. In the case of a soccer game this can be avoided by a rule that only the Calculator node of that client which has shot the ball can calculate the trajectory of the ball.

3.4 Naming EventStreamSensors

For more complex scenes, EventStreamSensor contains a field SFString name, so that multiple EventStreamSensors, which may be located in different PROTOs can communicate events independently.

3.5 Simple Server Side Calculations

Besides the set_ prefix for eventIn fields, the EventStreamSensor also allows other prefixes. These trigger simple server side calculations. As an example, if an event is sent to toggle_DoorOpen, the SFBool value DoorOpen will be toggled, and the result will be sent to all clients via the DoorOpen_changed eventOut field. This way, the touchTime of a TouchSensor connected with the door geometry can be sent directly to the toggle_DoorOpen state of an EventStreamSensor and the DoorOpen_changed eventOut field can be sent to a Script node that updates the door state in all clients. The most common prefixes are add_ and dec_ for most SF values, inc_ and dec_ for SFInt32 values, and append_ for MF values.

3.6 States versus Events

BS Collaborate makes a distinction between state variables and events. State values are values that may change over time, and a user joining the scene needs an update of the current state of all these values.

State variables are stored by the server and the server can do calculations like toggling an SFBool, or incrementing an SFInt32. Currently states are stored in memory and are not preserved when the server is restarted, but a later version may store them in a data base. For states it is not necessary that every update comming from a client is forwarded directly to all other clients, only the most recent value is important. Therefore a later version of the BS Collaborate server

may reduce bandwidth requirements by forwarding only a certain number of updates per second, which can be specified by the content author.

Events, on the other hand, will just be forwarded verbatim by the server to the other clients. Events can be used for things like synchronizing which client is allowed to calculate an animation that is then distributed to all other clients.

State values are sent to the server via the set_ prefix, and received from the server via the _changed postfix. They can be modified via prefixes like toggle_ or add_. Events are sent via the evt_ prefix and are received via the _evt postfix.

4 Connection to the server with Connection node

The *Connection* node is responsible for the connection to the multi user server.

```
NetConnection
{
   field       SFBool      enabled     TRUE
   eventOut    SFBool      isActive
   field       MFString    address     "localhost"
   field       SFInt32     port        0
   field       SFInt32     protocol    0
   field       SFTime      timeOut     0
   field       SFBool      secure      TRUE
}
```

With the field *enabled* you can activate or deactivate the connection to the server. If a connection to the server has been established it is indicated through the field *isActive*.

The server address is defined in the field *address* and the including port number for the connection is specified in the field *port*.

The field *protocol* indicates which version of the protocol controls the communication between server and client. The list below comprises the available protocols.

protocol number	Protocol name
1	HTTP
2	TCP/IP
3	UDP/IP
4	BSMUP (BS Multi User Protocol)

The fields *TimeOut* and *secure* are not implemented now and will come later.

5 Avatar position and chat messages with BSCollaborate

5 Avatar position and chat messages with BSCollaborate

The node BSCollaborate is a node that handles the user login and logout. It is also managing the position of each connected user. This node is managing the chat communication between the users as well.

```
BSCollaborate
{
        field       SFNode      connection
        eventIn     MFString    tryLogin
        eventOut    SFBool      loginResult
        eventIn     SFTime      logOut
        eventIn     SFVec3f     userPos
        eventIn     SFRotation  userOri
        field       MFNode      users   []
        eventOut    SFNode      hasJoined
        eventOut    SFNode      hasLeft
        eventOut    SFNode      hasMoved
        eventIn     MFString    meSay
        eventOut    SFNode      hasSaid
}
```

The field *connection* must point to a connection node. See an example in chapter 5.1.

With the MFString field *tryLogin* a user can connect to the specified server in the referenced node *connection*. The first element of the *tryLogin* field is the login name and the second element is the password.
tryLogin=('loginname','password','')

The login result is available in the field *loginResult*. If the the login was successful the value of the field is TRUE and the field *tryLogin* will be ignored. If the value is FALSE another try is possible.

The two fields *userPos* and *userOri* are events to the server from the own position and orientation of the avatar. You can send these events using a route from a *ProximitySensor*. The code below is an example to show how to calculate the user position.

```
DEF Proxi ProximitySensor
{
        size 1e30 1e30 1e30
}
ROUTE Proxi.position_changed TO BSCollaborarte.userPos
ROUTE Proxi.orientation_changed      TO BSCollaborarter.userOri
```

The MFNode field *users* contains information about each logged-in user. The first element in the list is always your own user. The structure of the node is

```
UserData
{
        field SFInt32       idx          -1
        field SFString      nickname      ""
        field SFString      avatarString  ""

        eventOut SFString   loginState

        eventOut SFVec3f    posistion
        eventOut SFRotation orientation
        eventOut SFBool     isMoving

        eventOut MFString   chat

        field    SFNode     userData     NULL
}
```

The available states of *loginState* are:

joined:	Only for other users.
logged-in:	Only for the own user.

With the eventOut *hasJoined* the server is sending the information of the user which has joined the server. The node structure is the same as the one of the node *users*.

The node *hasLeft* is useful to handle users which are logged-off from the server. The node contains useful information about the those users.

With the node *hasMoved* you can identify which avatar is moving. The node structure is again the same as in the node *users*.

For chat implementation you can use the fields *meSay* and *hasSaid*. With the field *meSay* you can send a text message to the server. If the message has been successfully delivered to the server, it sends back the message to the field *hasSaid*. The node structure of *hasSaid* is like *users*. After receiving the same message from the server you can be sure that the message was delivered to other users in this virtual world.

6 Shared Events available with EventStreamSensor

With the node *EventStreamSensor* you can create shared events between the users and the server. This means you can manipulate virtual objects in a collaborative environment. This node receives the changed field values from server and is sending new field values to the server.

```
EventStreamSensor
{
        field       SFNode      connection
        eventOut    SFBool      initialized

        fields
        ...
}
```

The field *connection* must point to a connection node. All events will communicate with the server address in this node.

With the field *initialized* you can make sure that all initial values are received and the server has finished sending.

Each *EventStreamSensor* can handle an unlimited number of different fields. Each variable has a prefix or a suffix. The following list of available pre-/suffix explains their functionality.

List of prefix

- set_ : set the value
- add_ : add the specific value to the variable. Result value is stored on server.
- inc_ : increase the number by one. Result value is stored on server.
- sub_ : decrease the number by one. Result value is stored on server.
- evt_ : unstore event to the server and the server is sending this event to all clients

List of suffix

- _changed : Indicates a stored event from server
- _evt : This value comes from the server and was not stored.

A list of variable types and their available prefixes:

SFBool	set_ toggle_(SFTime) setTrue_(SFTime) setFalse(SFTime) and_(SFBool) or_(SFBool) inh_(SFBool) xor_(SFBool) equ_(SFBool)
SFColor	set_
SFColorRGB	set_
SFDouble	set_ add_ sub_
SFFloat	set_ add_ sub_
SFImage	set_

SFInt32	set_ add_ sub_ inc_(SFTime/SFBool) dec_(SFTime/SFBool)
SFNode	set_
SFRotation	set_ add_ sub_ (is more a multiplication and a inverse- multiplication operation)
SFString	set_ cat_
SFTime	set_ add_ sub_
SFVec2f	set_ add_ sub_
SFVec2d	set_ add_ sub_
SFVec3f	set_ add_ sub_
SFVec3d	set_ add_ sub_
SFVec4f	set_ add_ sub_
SFVec4d	set_ add_ sub_

MF fields use the same methods as SF fields, plus an append_, which accepts an SF value or an MF value. If an MF field receives an SF via set_, their type stays MF, but the NumberOfElements becomes 1. If an MF field receives an SF via a method other than set_ and other than append_, then the value influences all the elements of the current value, i.e. NumberOfElements does not change. If an MF field receives an MF via a method other than set_ and other than append_, then the current value is extended if the received value has more elements, but is not truncated if the received value has less. The operation however influences only the first N elements of the current value, with N == NumberOfElements of the receive value. This applies only if no other mode makes more sense for the given operation. If an MF field receives an MF via set_, then the current value is set to the receive value, and its NumberOfElements changes to the one of the receive value.

MFBool	set_ append_(SFBool or MFBool)
MFColor	set_ append_(SFColor or MFColor)
MFColorRGB	set_ append_(SFColorRGB or MFColorRGB)
MFDouble	set_ append_(SFDouble or MFDouble)
MFFloat	set_ append_(SFFLoat or MFFLoat)
MFImage	set_ append_(SFImage or MFImage)
MFInt32	set_ append_(SFInt32 or MFInt32)
MFNode	set_ append_(SFNode or MFNode)
MFRotation	set_ append_(SFRotation or MFRotation)
MFString	set_ append_(SFString or MFString)
MFTime	set_ append_(SFTime or MFTime)
MFVec2d	set_ append_(SFVec2d or MFVec2d)
MFVec2f	set_ append_(SFVec2f or MFVec2f)
MFVec3d	set_ append_(SFVec3d or MFVec3d)
MFVec3f	set_ append_(SFVec3f or MFVec3f)

In the following *EventStreamSensor* example you can find the notation of this sensor.

```
EventStreamSensor
{
        field       SFNode      connection IS NetConn
        eventOut    SFBool      initialized

        eventIn     SFVec3f     set_BallDestPos         #stored
        eventOut    SFVec3f     BallDestPos_changed

        eventIn     SFBool      evt_ResetBall           #not stored
        eventOut    SFBool      ResetBall_evt

        eventIn     SFTime      inc_GoalA               #stored
        eventOut    SFInt32     GoalA_changed
}
```

7 BS Collaborate server

The Bitmanagement Software Collaborate Server was built to handle multiple connections from the BS Contact 7.1 clients. Its task is to manage the login requests, user avatars and events from the connected clients. In order to store persistent events the server has the ability to connect to different ODBC SQL databases or write the events in a file system as a fall back. An existing user databases can be used as the database that stores user accounts. This can be configured in the configuration file. With this file you can customize the server and adjust proprieties to your needs. You can expand the functionality of the server by writing your own server side script. These scripts can compute various and add functionality to the multi-user environment.

8 Server configuration file

8 Server configuration file

The server configuration file is a human readable text file. It will be read and parsed at the start up of the server. With this file you are able to configure and customize the server in various ways. The following configurations are possible.

In TCP/IP and UDP networks, a port is an endpoint to a logical connection. Some ports have numbers that are preassigned to them by the IANA. These ports are known as well known ports (specified in RFC 1700). In the range from 0 to 1024 are the reserved port numbers for privileged services. That is why it is recommended that you specify a port number for the BS Collaborate server above 1024. To define this number you can assign it to this variable

BS_CollaborateServer.port = [port number]

Example:
BS_CollaborateServer.port =12345

your preferred and currently not occupied port.

8.1 Storing events with database

In order to store persistent events from the event stream sensors you can define in the system how these events will be stored. The recommend way is to use a SQL database. To get the server working together with the database you have to install ODBC and the specific ODBC driver for your database. Configure the DNS with user name and password from the database privileged user and location of the database server. To use the SQL database as the storage system, assign the value 'Database' to the variable *storage.system:*

storage.system = DataBase

To create a connection to the defined database you have to enter your DNS name after:

storage.params.dsn = [DSN Name]
Example:
storage.system = MySQL

The database system creates a number of databases and tables. Every scene will be stored in a separate database and every *EventStreamSensor* node identified by *streamName* in this scene is a table in the corresponding database. This means you can manage a large number of different scenes and *EventStreamSensors* on the server without creating conflicts.

8.2 Storing events with a file system

You are also able to store events in files. This means that every *EventStreamSensor* node will be one file in a special directory hierarchy. To use this storage system you have to use the following parameter:

storage.system = FileSystem

To define the directory for the storage of the events you have to enter your preferred path as value. You have to work with double backslashes in the path.

storage.params.path = [directory path]
Example:
storage.params.path = "C:\\BS_Collaborate"

This storage system creates a separate folder for each scene. In these folders every *EventStreamSensor* is stored as file. You can have many different scenes and *EventStreamSensor* without creating conflicts.

8.3 Summary of the configuration file

BS_CollaborateServer.port =	*[port number]*	*0-65535*
storage.system=	*[DataBase, FileSystem]*	
storage.params.dsn =	*[DSN name]*	
storage.params.path =	*[directory path]*	

9 System Requirement

Internet connection: 56 kbit/s

Client Operation system: Windows 98 SE, Me, 2000, XP und Vista

Server Operation system: Windows 2000, XP und Vista, (embedded) Linux, HP-UX, Tru64, Solaris, QNX

Graphic standards: OpenGL 1.1/1.2 and Direct3D (DirectX) 7/8/9

10 Download

http://www.bitmanagement.com

BS Contact 7.1 Client:

http://www.bitmanagement.de/download/playerdownload.en.html#BS_Contact_VRML/X3D

BS Collaborate:

http://www.bitmanagement.de/download/playerdownload.en.html#BS_Collaborate

Tutorial for developing multi user scenes and applications

http://www.bitmanagement.de/developer/collaborate/tutorials/index.html