DimensioneX version 7.0
Developer's Reference

by Cristiano Leoni

Last-updated: 28 January 2023

Index

1 Basics

1.1 Important notices

Before starting reading this documentation you are assumed to have DimensioneX installed and correctly configured to run. All the information required to do this is in the **Readme** file included in the DimensioneX kit. Please refer to that file also for legal/licensing issues.

This documentation is designed to be your main reference when developing a game on DimensioneX, it is kept updated thorughout time as features are added to the engine.

1.1.1 Tutorial

Before diving into this documentation, It is recommended that you follow a short tutorial "How to develop a multiplayer game with DimensioneX". You can find it on-line on the DimensioneX official web site, just click the "**Tutorial**" link.

1.1.2 Support

You can get **free help** when developing on DimensioneX: Just go to the DimensioneX web site and go to the ***SUPPORT*** section.

1.1.3 Send us your work!

You are encouraged to send us your work or at least send us some lines about your project.
We offer free hosting for any games developed with DimensioneX, while giving full credit to the authors. We also offer free links and free visibility to your project site if you want to host it by yourself.
You can join the DimensioneX project and receive our newsletter: just go to our website, click on the "***JOIN!***" button.

1.1.4 Contributions

It is extremely difficult to keep a project up a running with many users asking for support and help, and no one giving any contributions. DimensioneX is self-supported and you can contribute in a few ways:

- **Making new games** and releasing them / letting us know – This will attract new visitors and more resources for the project
- **Spreading the word** – write papers, articles, posts or link to us. Also, please let us know. If you use DimensioneX for a research work and we don't know it, our references will remain unchanged: you also would have missed an opportunity to gain more visibility.
- **Donating money** – a secure system via PayPal and SourceForge.net lets you offer with no risk. A donation as little as 5$ is a good encouragement for us and will help the project to stay alive for more.

You can learn more on DimensioneX's project by participating to the discussion on our social channels.

Sometimes users ask for new features and modifications to the game engine without giving contributions or under the generic promise to do something later, once their requests have been satisfied: **you should not expect to achieve much his way**.
Once you have contributed, *then* you can ask and have chances to be satisfied.

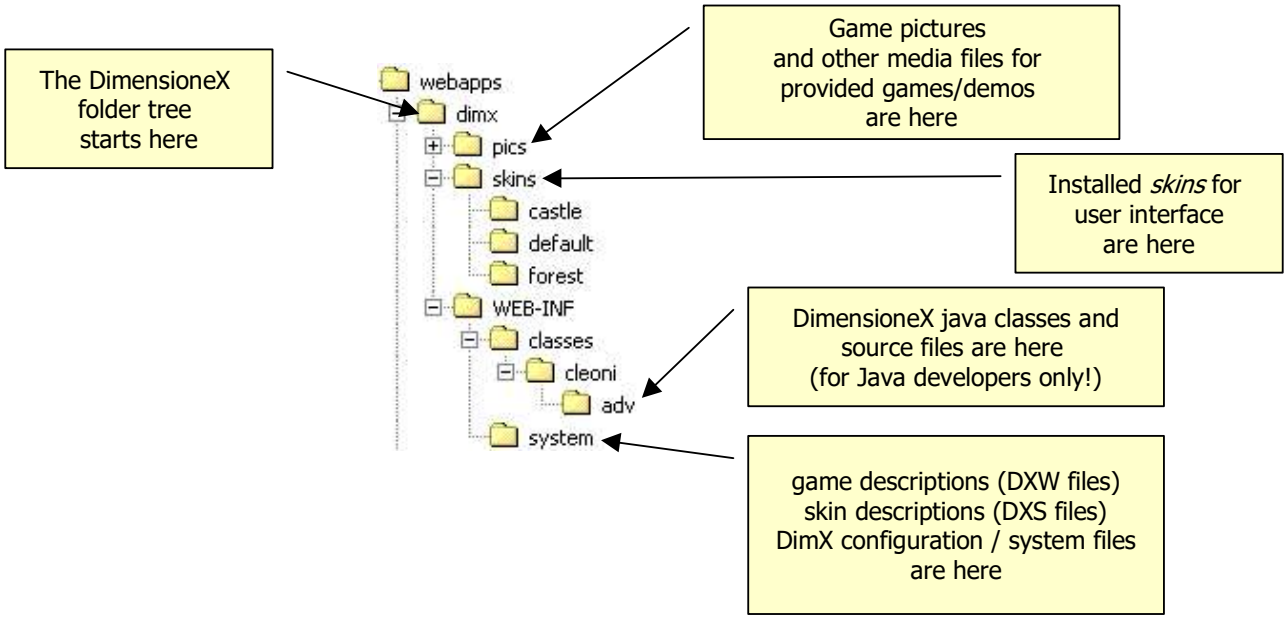
1.2 How DimensioneX works

- ✓ **The DimensioneX engine is a Java Servlet.** This servlet is named *multiplayer* and runs inside a web server (typically Tomcat).
- ✓ **The game engine** knows how your game is composed (rooms, characters, items) and keeps track of all the changes in the game. It reads the game description which is stored in a DXW file (actually a plain text file) and rebuilds it into its interal memory.
- ✓ **The users** connect to the game in the same way they are accessing an interactive web site. As the webserver sees the first user, it loads the *multiplayer* servlet, reckognizes the game slot that the user has specified on the url (game=*N*) then reads a configuration file (worldnav*N*.properties where N is the slot number). Then, the game file is loaded and the game world is constructed into the game engine's memory. Since then, the game engine keeps track of the user's status and position, and makes all the interactions to happen in the game.
- ✓ The game engine outputs HTML and Javascript to the user browser. Also it uses things named *cookies*. This all constitutes the game's visual interface and can be customised by the game developer in several degrees. Please note that this guide is not focused on this topic. If you find this guide incomplete and think that your game is OK already but the user interface needs more work, then contact the author.

1.2.1 Folder structure - What goes where

Under your webserver application folder (Usually **C:\tomcat\webapps**) there should be your DimensioneX folder, named **dimx**.

Below you see the folder structure of a typical DimensioneX installation:



The most important folders for you to consider are:

dimx	this is where the dimx web tree starts.
system	We call it the DimensioneX *system folder*. This is where you should keep: - games definitions (.DXW) - skin definitions (.DXS), - configuration files (worldnav*N*.properties) - language files (msgs_xxx.properties) - debug files (.LOG – automatically generated by DimX). - player profiles files (gamename.SAV – automatically generated by DimX) - hall of fame file (gamename.HOF – automatically generated by DimX) - HTML templates (.HTM, .HTML) Please note that you will find here the standard log file, debug.log. Creating a shortcut on your desktop for quick access to this folder is a good idea.
pics	this is where the graphics for demo games (such as The Beach) are kept. Typically you will create separate, parallel folders for your own games so that files do not get mixed up. Any folder for this purpose can have subfolders (eg. if you like to keep separated music/sound files from graphics).
skins	This is where files for skins are installed. One folder per skin: in the picture above you can see there are three installed: default, castle and forest. The game engine, however, assumes that skin's DXS file (skin definition) is however located in the system folder. The skin folder name must match the skin's ID
classes	This is where the DimensioneX game engine classes are kept. In the cleoni/adv folder you will find also the java sources.

Remember: for each game you develop, you should create a separate folder where to keep the game pictures and media files.

1.2.2 Custom command panels

Available commands are determined by sets of controls named PANELs.

The game engine has a pre-defined set of available commands which produce a series of a well known events and actions. The game programmer, though, can produce custom command panels and use them in the game, as well as she can define custom events to be triggred when custom controls are used.

Command PANELs can be assigned to players and to rooms via the **PANEL** tag in the game's design section, or via the **SetPanel** instruction at run time.

During gameplay, the player is always located in a room. The available, displayed commands will be a combination of commands associated to the player and commands associated to the room:

```
Available commands = commands from player's panel + commands from room's specific panel
```

Check out the following sections for more information:

- DimensioneX DXW Reference -> PANEL definition
- DimensioneX DXW Reference -> ROOM
- DimensioneX SmallBasic Reference -> Instructions -> SetPanel

See also the Panels Demo included in the DimensioneX kit.

The panels used in login/logout sequences can be customized, too, but in the present version you have some limitations regarding custom scripts to be executed during these sequences. See "Game connection sequence" below for more details.

1.2.3 Dynamic pages and descriptions

Although DimensioneX is not designed to work as a dynamic website platform, in a number of cases you want a description of a room or object to dynamic.
The description property of each object can be made dynamic by placing a traling '@' sign in front of a function call, example:

```
ROOM xyz
DESCRIPTION @myFunction($AGENT)
```

Will produce a dynamic description by executing and using the return value of myFunction($AGENT). This is very handy for displaying context-menus associated to rooms and objects, without the need to using system events.

Since each room you are into displays a web page on the client, this gives the game designer the freedom to use DimensioneX as it was a kind of dynamic site platform. It is important to bear in mind,

though, that dynamic descriptions force the game engine to run code each time a user looks somehow at that specific room and object, and therefore they consume lots of CPU time.
Using system events instead of dynamic descriptions can, in most cases, improve your game performance.

1.2.4 Game connection sequence

The sequence for login/logout/save and exit/restore saved game (both panels and logic behind) are built-in the game engine and ready to use.

Recently an increasing number of users expressed the need to customise the logon sequence.

In the present release, all panels can be customised so the game programmer can easily add a field to any of the panels and remove existing ones, just be re-definining the correct panel via its panel ID (See the section: DimensioneX DXW Reference -> PANEL definition for more details).

The logic behind the panels, though, cannot be customised in the present version. In other words, if we add a new field for the player's e-mail there is no software hook to let us specify what to do with such piece of information.

For this reason, it is recommended that game developers use the standard sequences at present. They can, however, add special, custom ROOMs and write the proper scripts to add steps for players entering the game, where we let them specify their e-mail address, etc.

Experienced developers can modify the java sources to alter the game engine behaviour if the solution proposed above don't fit their purposes.

1.2.5 TYPE system

For the DimensioneX game engine, all ITEMs in the game are intrinsically the same type. The same applies for characters/players.
Nevertheless, it is useful in the game to have a means to distinguish between similar objects.
In true OOP languages, the programmer is forced to plan in advance and design a class hierarchy so fitting the program objects.

In the DimensioneX game scripts (SmallBasic language), there are some devices to simulate what OOP programs do. These tools are:

- The TYPE property
- Attached events

The TYPE property is some string with the form:

type.subtype

For example, the following definition:

```
ITEM sword1
TYPE sword.steel
```

```
ITEM sword2
TYPE sword.iron
```

Tells the game engine we are defining two items of type="sword" which slightly different as the subtype tells us.

If, at a certain point in the game, we need to detect swords in the room, we can use the **getObjectType** function like this:

```
iron_swords = getObjectsType($WORLD,"sword.iron") ' only those made of iron
all_swords = getObjectsType($WORLD,"sword") ' all swords
```

Also, we can use the AttachEvent instruction to tell the game engine that the steel sword has a slightly different effect than the iron sword has.

```
AttachEvent sword1,"onUse","steel_onUse"
AttachEvent sword2,"onUse","iron_onUse"
```

Upon clicking USE and any of the two swords, they will behave differently.

1.2.6 MP3 support

DimensioneX supports playing MP3 backing music on the player's browser by using modern HTML's native embedding technology.
The user just needs to enable the "Music" option on the login screen, and enable pop-ups when on the game server.
To learn more, see the PlayBackground instruction, and the section about server settings (dimensionex.properties): protectMusic setting.

1.2.7 Custom client behaviour

The behaviour on the client (user's browser) is determined by a combination of the following elements:

- **Frame structure**: contained in the client file (file with ".client" extension on the server, as configured in the worldnav.properties file)
- **Client-side scripts**: Javascript contained in the script file (file named "client.script" on the server)
- **Appearence**: User chosen skin (DXS file on the server, possible choices listed in the game's GUI section)
- **Messages:** Language file in use (as configured in the worldnav.propertis file)

With some lower probability, client behaviour can be determined by:

- Command PANELs in use
- HTML and simple javascripts produced by the game engine
- HTML and scripts produced (typically via the Print instruction) in the game scripts

The game programmer can influence all the elements, with the sole exception of what is generated by the game engine (built-in) which can only be altered by modifying the game engine java source. The

important step is to figure out what needs to be modified to produce the desired effect. A little trial and error will help, but you can ask for help in our Support section.

To change the client behaviour, some knowledge of HTML, CSS and Javascript is required.

Recommended reading in this document:

- Basics->Configuration
- SKINS->Customizing the user interface
- PANELs
- About the DimensioneX Client

1.2.8 Stopwords system

To limit the risk of people being offended by nerds who log in and use bad wording in the game, the game engine offers a filter that blocks unwanted characters and words.

- The filter can be configured at server slot level (see Basics->Configuration ->Game Settings, worldnav.properties file) and it is active on nicknames.
- To activate the filter on the chat system, you simply state this in the WORLD definition by using the MUTING tag (see DimensioneX DXW Reference->WORLD). You can even fine tune the way the filter will work on the chat.

1.2.9 Banning system

Due to persistent presence of nerds who enter games and offend, the game engine was equipped by an effective banning system.
The banning of a user can be done in two ways:

- At run-time by means of a script, by using the **Ban** instruction. This method is particularly effective as it provides what we call *sticky* banning.
- At run-time by altering the WORLD's property named "_bannedClients". This is actually a "black list" containing all banned IPs separated by pipe signs.

The banning of a user can be permanent or not, at game programmer's will. All you need is to save banned users' IP numbers (_bannedClients property) on disk by using the **saveSetting** instruction, and restore it back by using the **getSetting** instruction when the game restarts.

The banning of a user can be *sticky*. When a user is banned via the Ban instruction, a cookie is silently passed to the user's browser. If the user changes IP and tries to log back in, he is reckognized because of that cookie, and the new IP is added to the black list.

When a user is banned, he cannot log on any more on any of the slots of the server on which he has been banned from. At each trial, he will be redirected to a standard page telling him he has been banned: http://dimensionex.sourceforge.net/banning.htm

To allow back a banned user, both the following conditions must be verified, in the provided order:

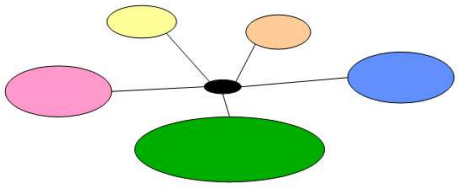
1. The users uses an IP which is not in the game's black list (_bannedClients property)
2. The users has cleared its cache *and* cookies before attempting to log on.

For more details on how to use this feature, see the following sections:

- Basics->Configuration ->Game Settings
- DimensioneX SmallBasic Reference->Object Models->World->_BannedClients
- DimensioneX SmallBasic Reference->Instructions->Ban

1.2.10 Multi-Area support (Clustering)

DimensioneX offers you the possibility to build a complex game by combining several independent areas into CLUSTERs of worlds.



Worlds belonging to the same CLUSTER must declare it in their WORLD definition. From then on, characters and objects can be moved between areas by using the **MoveOutside** instruction.

Clustered worlds do not share a memory space, even though they will use the same settings database (named after the cluster). Information to be shared can be safely passed to other areas by attaching it as a property to dummy message objects, which can be detected and destroyed after the information has been received.

For more details on how to use the clustering feature, see the following sections:

- DimensioneX DXW Reference->WORLD->CLUSTER
- DimensioneX SmallBasic Reference->Instructions->MoveOutside
- Basics->Developing a Game->Developing Multi-Area Games

1.2.11 Persistence

Game persistence is the property your game has to *persist* features, such as to let the world envirnoment to be the same after a server restart, and to let the user retain her objects after she logs off and in again.

DimensioneX provides to the game programmer a number of facilities to ensure game persistence. We divide these facilities according to what you need to make persistent: **world envinronment/context** and **user items**.

1.2.11.1 World environment/context

The persistence of the game status is not guaranteed automatically by the system, so the game programmer must ensure that the relevant game status information are saved at regular basis and restored at restardt, by wisely using the **saveSetting/getSetting** instructions:

- By using the **onTick** event, possibly every N ticks, you can save relevant context information via the **saveSetting** instruction.
- By using the **onStart** event, you can read back the saved information via the **getSetting()** function, and reconstruct the relevant context.

Warning!: The **saveSetting** instruction produces a disk access and therefore quickly degrades game performance. For this reason, it is recommended to:

- use it only for *really relevant* information
- use some programming logic to save game status at every N ticks (where n can be equal to 4 or more) in order to decrease performance degradation.

1.2.11.2 User items

Since user properties are saved by the system each time the user executes the SAVE or the SAVE & EXIT commands, the persistence issue only affects the owned items.

DimensioneX simulates real life so, when a user logs off, the owned items drop to the floor and can be used by other characters. This behaviour is drastically different to that of single-user games, where everything is related to the only user and everything can be destroyed and reconstructed together with the user.
Nevertheless, DimensioneX helps users to have their items back along three levels of restore:

1. Save and Restore via the object's **unique ID**.
   This system is fully automatic.
   How it works: When the game is saved, all owned object IDs are saved. When the user logs back in, she will re-obtain the same objects IF those objects are still unused by other characters and are not locked somewhere in container objects (closets, etc.), nor any event cancels this restore operation.
2. Save and Restore via the object's **type** (*AutoRestore*).
   This system is fully automatic provided the game designer has consistently used the object **type** property to specify object types. This system can be disabled.
   How it works: When the game is saved, all owned object **type**s are saved. When the user logs back in, if the restore based on unique ID fails (see above), then the system searches for a free object having a **type** matching that of the saved item. If also this search fails, it will search for an object with at least the same **main type**.
   Example: John owns an iron sword. He saves and exits, then logs back in. His sword is in use by another character now so the system searches for another sword of the same type ("sword.iron"). If this search fails he will search for objects of the same main type (eg. "sword.silver", "sword.simple" will then match). If a positive match is found, the object is assigned to John.
3. **Serialization/Deserialization**.
   This system must be implemented by the programmer via two scripts: saveInfo( ) and restore( ) and it is similar to what happens in other languages (Java) for similar needs.
   For this system to work you should have implemented the following:
   a. A function that is capable of building objects starting from their **type** and, optionally, extra information that can change from different object types
   b. A *serializing* **saveInfo**( ) function that, based on the object's type, can extract relevant information and serialize them into a string that is returned as a result. This string will be saved along with object's type for later restore.
   c. A *deserializing* **restore**( ) function that, taking as input the desired object type and the saveInfo string generated by saveInfo( ) function, can *deserialize* the object and thus reconstruct it, possibly calling the item-creation function mentioned at the point a).

The following rules apply for **game save**:

- All players' properties are saved in the player profiles' database, under the key: "username_properties". References to objects are converted in object IDs
- All objects ain inventory are saved in the player profiles' database, under the key: "username_items".
- For each object, the ID is saved.
- If the object has a TYPE, the TYPE is saved, too.
- If the user has performed SAVE & EXIT, object has a **saveInfo**( ) event attached, this EVENT is called. If its result is different than NULL, then it is saved.

The following rules apply for **game restore**:

- If a **restore()** event was defined, it is called with the object information which is contained in the saved game profile. The **restore**() event is supposed to restore the object by using the provided information. If the **restore**() event's result is **true**, the system clears the saveInfo information it has in the saved profile and advances to the next object in the list, if not, the system attempts to use the Auto-Restore feature.
- For performing Auto-Restore on an object:
    - the system tries to search current world for an object with the same ID.
    - If TYPE information was saved, it will check against object TYPE too.
    - If a match is found, and the object can be moved, it is reassigned back to the player's inventory.
    - If a match cannot be found, then the current world is searched for a free, moveable object with the same TYPE. If the operation succeds, the object is assigned to the player's inventory.
    - If a match cannot be found, the system gives up and a message "cannot restore…" (or equivalent) is displayed.
    - The system then advances to the next object in the list.
- The Auto-Restore feature can be disabled. Use the worldnav.properties setting **disableAutoRestore** for this purpose.

1.2.12 Server-Side Hooks

DimensioneX offers a way to integrate any game with external applications.

You can use the HOOKS tag to:

- Export game data as an XML feed
- Integrate your game with external on-line shopping carts or e-commerce/e-payment procedures
- Integrate your game with Google Toolbar Buttons
- Integrate your game with virtually ANY web-based application

HOOKS defines commands that the game server will accept from the outside, regardless of the state of the client (in-game player or not). Each command you specify in the HOOKS line will be associated to an event defined by you. Then, in the event code, it will be up to you what to process and what to output.

1.2.13 mySQL Support

Game settings are normally saved on disk in text files with .sav extension. In production servers, performance can be improved by using a mySQL database instead.

DimensioneX is capable of using a mySQL database for storing data transparently to the user. The connection is automatically created for each use and, if a failure occurs, the game engine will switch to ordinary file-system based behaviour hence allowing the users to continue playing. Specific events (onDbDown/onDbUp) will notify your game about this so that you can temporary disable save game function, inform users, etc.

To activate mySQL support, you just have to:

- Create an empty database on any mySQL server
- Configure the dimensionex server to be aware of it. You must configure credentials for a user that is capable of creating tables and writing to them.
- Run the **Check Database / Initialize** function from the **Maintenance Panel** – See section about Maintenance Panel below for more details.

The Maintenance page offers utilities:

- To check database connection and diagnose problems. This procedure also initializes the tables for their first use: A table for each game is automatically created, and values are automatically loaded by taking them from the existing .SAV file, if any.
- To easily load data from the SAV file into the mySQL database, and the converse export operation.

It is recommended, though, that the server admin uses some other specific tool (such as phpMyAdmin) to back up the database and/or backup its content in production installations.

1.2.14 Output to RSS feed – integrating with Facebook, Twitter and other social networks

DimensioneX offers a way to publish the most important in-game events to a standard RSS feed. From there, this information can be easily distributed to the interested audience, either via standard RSS feed readers and browsers, or via Facebook, Twitter and any social networks.

- Piping an RSS feed to a Facebook or Twitter account, or to a Facebook page is easy by using available free services such as Twitterfeed.com and Hootsuite.com. The use of such services is beyond the scope of this manual.

In order to produce this RSS feed, DimensioneX lets you produce a Journal file which is actually a text file in which events are listed recent-to-old order, one per line.
The journal file which is managed by the system to list the 20 most recent events, is a pipe-separated plain text file. An example of journal file produced by DimensioneX follows:

```
Thu, 26 May 2011 12:56:24 +0200|Cris rules the Kingdom!|http://www.underworld-
game.net/infobox.php?id=cris&amp;lang=eng|There is a new King in Underworld:\n<a
href="http://www.underworld-game.net/infobox.php?id=cris&amp;lang=eng"
target=_blank>Cris</a>!|news,kingdom,user:Cris
```

```
Thu, 26 May 2011 00:39:13 +0200|Sir Duncan governs
Underworld|http://www.underworld-game.net/|There is a new King in Underworld:\n<a
href="http://www.underworld-game.net/infobox.php?id=sir_duncan&amp;lang=eng"
target=_blank>Sir Duncan</a>!|news,kingdom,user:Sir Duncan
Wed, 25 May 2011 14:54:08 +0200|a giant worm governs
Underworld|http://www.underworld-game.net/|There is a new King in Underworld:\n<a
href="http://www.underworld-game.net/infobox
```

As it stands, the **journal file is not a standard RSS file**. The management of the text and of RSS format would slow down the execution of the dimensioneX server and therefore it has been designed to be done externally.

Then, the game developer must implement its own Journal-to-RSS feed converter, possibly on the game's official website.

On the DimensioneX website, Downloads section, we provide a free, Open Source, Journal-To-RSS converter which works in real time written in PHP which is perfect for general use.

The Journal file is tied to the game's Cluster (see also: **1.2.10 - Multi-Area support (Clustering)**) and can be configured inside the worldnav.properties of any of the cluster's worlds (see also: ***1.3.3 - Game Settings***).

```
journal=C:/tomcat/webapps/dimx/journal.txt
```

We recommend to generate the Journal file in a folder that's accessible from the web. In the above example, the journal.txt file is generated so that it can be accessed by http://localhost:8080/dimx/journal.txt in a standard development environment. Please adapt this logic to your own case.

See a working implementation of this system on our Underworld Online game, running on DimensioneX, at http://www.underworld-game.net . The journal file is generated in a public folder of the dimensionex server at Gamesclan.it (we will not disclose the exact URL, but it's not important) then the official website fetches it and turns it into a standard RSS feed in real time. Then free services pipe these events to the game's Twitter channel and Facebook page.

1.3 Configuration

1.3.1 System folder

All the game engine settings are kept in the *DimensioneX's system folder*.
This folder is usually located on your server, and it is something like the following:

tomcat/webapps/dimx/WEB.INF/system

As it stands, the system folder is located inside the webserver's (Tomcat's) tree.

Notice: The dimensionex system folder location have been designed to be un-accessible by the internet users.

Normally, all the content inside the WEB-INF folder is kept hidden and it cannot be accessed. If yours is a public installation, please verify that this URL does bring a NOT FOUND error:

http://yourserver/dimx/WEB-INF/system/dimensionex.properties

If the content of dimensionex.properties is visible, instead, then you have to find an alternate, suitable place to use a system folder.

1.3.1.1 Using an alternate system folder

You have to follow this procedure in case your system folder contents is accessible via a web browser.

1. Identify a folder that is outside the web content tree, and which is still accessible and writable by you
2. Configure the **web.xml** file in your DimensioneX installation (WEB-INF folder) so that it contains the following **base** parameter configuration (notice: this example works for Tomcat 4.1 only – for different tomcat versions, please check out Tomcat docs):

```
<servlet>
  <servlet-name>multiplayer</servlet-name>
  <servlet-class>cleoni.adv.multiplayer</servlet-class>

       <init-param>
          <param-name>base</param-name>
          <param-value>/path/to/custom/dimx_system/</param-value>
       </init-param>
</servlet>
```

3. Restart the server. Try accessing a DimensioneX game and check out its debug.log
4. Verify from the first log messages that the system is not "guessing" the system folder, but it is using the "base" setting you have just placed in the web.xml file. (If it does not read the setting, consult Tomcat documentation for info about how to store config parameters, and repeat)
5. Verify that, when connected to a test game, you can do a [Save Game], that is: the game engine should be enabled to write into your moved system folder. If not, give proper access rights for folder reading, opening (execute access on the folder) and writing (in a few cases you have to make the folder world-writable). Consult your server administrator in case of doubts.

1.3.2 Server Settings

There is a configuration file named **dimensionex.propeties** whose settings affect ALL game slots. This configuration file determines how the whole DimensioneX server will work.
To configure your DimensioneX server, open with a text editor the file: **dimensionex.properties** located in the **tomcat/webapps/dimx/WEB.INF/system** folder.

Modify these settings only when you have some confidence with DimensioneX.

adminPasswd	Specifies the server's administrator password. This can be requested for some special actions in the the maintenance page.
serverType	Possible values*: public | local

	Specifies whether yours is a **public** or **local** server. On the basis of this setting, the game engine will automatically choose the right IMAGESFOLDER for the loaded game (make sure you used IMAGESFOLDER_PUBLIC and IMAGESFOLDER_LOCAL). If it's a **local** server, the game's HTMLCOUNTER tag will be ignored. Default setting is **public** (standard behaviour)
savingDir	Folder to be used for saving game profiles and hall-of-fame files. By default this setting is (blank), so the base DimensioneX system folder will be used. Set it just in case you want to save game profiles into an alternate folder. Please use normal slashes "/" on Windows systems Notice: If you are using mySQL db (see below), the game profiles will be saved in it and this setting will be ignored.
msgRefreshRate	Number of seconds between refreshes of the chat window. Zero means "no refresh". Default value is 5 seconds
navigatorUrl	URL of the navigator DimensioneX servlet you specify it if automatic detection in your DimensioneX installation fails.
dbdriver	Java class of the JDBC driver for accessing the game profiles database. Default is: com.mysql.jdbc.Driver
dbhost	Host running a mySQL database for game profiles Default is (none). Usually you specify **localhost**
dbport	Not used yet, but it will be in future releases. It is the port at which the database service is accessible. Leave blank for standard port.
dbname	Name of the database which holds the game profiles.
dbuser	User ID capable of USAGE (SELECT, INSERT, UPDATE, DELETE) and CREATE TABLE on the specified database.
dbpass	Password of the above user

***Note:** The pipe sign (|) is used here to separate possible values but should **not** be typed in the configuration file.

1.3.3 Game Settings

There is a configuration file for each server slot, named **worldnav*N*.propeties**. This configuration file affects the way the game slot will work, and first of all **selects the game file** to be loaded at startup. To configure a game slot, open with a text editor the file: **worldnav*N*.properties** (N is the server slot to which the game should be associated). This file must be located in the **tomcat/webapps/dimx/WEB.INF/system** folder.

Note: The pipe sign (|) is used here to separate possible values but should **not** be typed in the configuration file.

Setting example	Meaning

`worldFile=World3.dxw`
 	
or, as well,
 	
`worldFile=http://www.domain.com/path/to/worldfile.dxw`
	Fundamental.
	Tells DimensioneX which is the file (that is, the game) to be loaded at startup. The game file should be placed in the same folder as the configuration file.
	
	Alternatively, if your PC is connected to the network, you can also specify an URL which will cause the DXW game file to be fetched and loaded automatically at startup.

`debugTo=`**`file`** `debugTo=`**`console`** `debugTo=`**`none`**	Tells where the engine output goes. **file**: If you specify "file" then the output goes to the file: debug.log so it is easier to understand where problems occurred. *Note: Beware, remember you have to delete the log file sometimes, otherwise it will waste more and more space!* **console**: the output goes to the console (small window on the server's screen) **none**: No debug information is produced, except for errors and explicit debug prints produced with the **Debug** instruction at run-tim. This setting allows the game to run at maximum speed and it is recommended for production or unattended servers.
`msgsFile=msgs_eng.properties`	Specifies the messages file. DimensioneX has already the messages file for several languages. If no file is specified, DimensioneX displays messages in English. You may edit the messages file for translation of messages and buttons in your own language.
`adminPasswd=`	Specifies the game's administrator password, the one requested by administrator page and the maintenance page.
`debugMode=`	If you set it to 1, you will have extra-fast login into the game (to be used as a time-saver when building or debugging games). Note: this setting can be detected in the game by means of the $WORLD.**debugmode** variable

`clientFile``=standard.client`
`|leftctrl.client`
`|bannerized.client`
	Specifies the client file to be used. This file is just a HTML template defining the client's frame. The default is structured in three frames, commands on the right, messages console with space for 3 messages.
	You may use another, or design your own and specify it here.
	Please note a 4-frame, banner-supporting client, is provided.

`clientScript``=client.script`	Specifies the client script file to be used. This file is

	supposed to contain a full SCRIPT tag to be inserted in the clientFile automatically by the engine.
journal=	Optional. Specifies a full path to a file in the local filesystem, this file will be used as a Journal file from the Journal instruction. See also Instructions->Journal or Producing RSS Feed
hideSourcePath=	Optional. In case of errors, the path to the game source will be shown by default. If you set this one to 1, game source path will never be shown to the users. This is recommended if the game source has to be protected.
analytics=<script src="http://www.google-analytics.com/urchin.js" type="text/javascript"></script><script type="text/javascript">_uacct = "UA-XXXXXX-1";_udn="none";_ulink=1;urchinTracker();</script>	Optional. You can use this setting to specify a full Google Analytics script which will be inserted in every display of the controls (ctrls) frame. Don't place line breaks (all on a single line). This setting will allow Google Analytics to "see" users spending time in your game and track the playing time with more accurateness. This is also believed to give your game a better Google positioning in the SERPs and a better view on players behaviour. Of course the ctrls frame should refresh sometimes for this to work well.
tracing=0 tracing=**1**	Optional. This setting works in cojunction with the server's database (see **Basics->Configuration->Server settings->dbname** in this document). If set to 1 (on), the game engine writes trace values in the database. This data is valuable statistic information for studying how the game is being used, e.g. for research purposes. Please note that the same information can be extracted from the debug text file via a regular expression text pattern. Default value is 0 (off, no tracing) If the database is not configured or not connected, this setting will be ignored.
disableAutoRestore=	Optional. When set to 1, disables the AutoRestore feature. Default value is null (AutoRestore enabled)
stopwords=<,>,*,fuck,idiot	Optional. Comma-separated list of forbidden words/symbols. Each "*stopword*" cannot be used inside nicknames (beware what you type, don't be too restrictive!). Also, the same words will be used for the WORLD.MUTING system (see) to avoid bad words to be used inside the chat system.

1.3.4 Local Language files

Most messages displayed by the game engine are taken from a language file named **msgs_eng.properties**. If you develop a game in local language, you will want to use or define a new local language file.
Please be sure to read also section *1.4.2 - Developing in local language (non-english)*, page 25.

1.3.4.1 Syntax

Language files are Java properties files. That is, they are expressed in the form

```
setting name= setting value
```

Empty lines are allowed, comments are marked with an initial "#" sign.
Here is a list of valid settings for language files.

1.3.4.2 Text Format

For these language files, both ANSI ASCII and UTF-8 text formats can be used. UTF-8 is the recommended one, although .properties files for Java programs normally need to be written using ANSI ASCII.

If you are defining a local language file, you should save this file in UTF-8 and write values as you would like to read them on the screen. If you find it more comfortable to use ANSI ASCII, then do so, but remember to set the encoding=ansi setting as explained below.

1.3.4.3 Settings

`charset`	(Mandatory) Specifies the charset to be used.
`encoding=ansi`	(Optional) Specify this line if the messages file is in ANSI ASCII text. Please note that all message values need to be encoded using the native2ascii utility, unless they all use letters of the plain english alphabet. If this setting is missing, the game engine assumes the properties file is saved with the UTF-8 format.
`displaymode=special`	(Optional) Tells the game engine to use an alternate, special display mode. Specify this line if the charatcters don't show up correctly in the browser. Please use it as a last-chance option (adjust charset first!) If this line is missing, the game engine uses the normal display mode.
`cmdNNN`	Defines text for game commands. NNN is a three-digit number such as 123,065,008 If missing, the english text is used.
`msgNNN`	Defines text for game commands.

	NNN is a three-digit number such as 123,065,008 If missing, the english text is used.

1.4 Developing a game

1.4.1 Recommended development environment

For developing a game, you need to set up a development environment composed of the following:
- DimensioneX game engine working on your PC.
- A good text editor, Textpad 4 (http://www.textpad.com is recommended)

1.4.2 Developing in local language (non-english)

DimensioneX is designed to support ALL languages and all charsets.
In the DimensioneX kit you will find already messages (XXXX_msgs.properties) files for several languages. These can be used with no effort from your part.

If your local language is not supported, then you just need to make a copy of the english messages, translate them, and configure the game engine to use your modified file. Please read this tutorial to have full details on how to do this:
http://www.dimensionex.net/en/docs/art012.htm
Be sure to read section *1.3.4 - Local Language files* at page 24 first.

The most important thing when you develop a game **using non-english characters** is that DimensioneX expects, by default, to load text files encoded with the **UTF-8 encoding** (although ANSI ASCII is supported – read on).

Please note that most text editors will use the ANSI ASCII encoding by default. So:

- Either you use a text editor capable of saving files in the UTF-8 format. Remember to choose UTF-8 when saving your game source files (DXW,DXL,TXT) the first time.
- Or you use the standard ANSI ASCII text format (most text editors in Europe and U.S. will be using it by default) but you need to specify ENCODING ANSI in your WORLD definition.

If you don't take care of this, non-english characters will not be displayed correctly.

European languages can be rendered both with ANSI and UTF-8. More complex charsets such as Chinese, Japanese, Russian, Arabian, etc. must be rendered by using UTF-8.

1.4.3 About game images and sounds

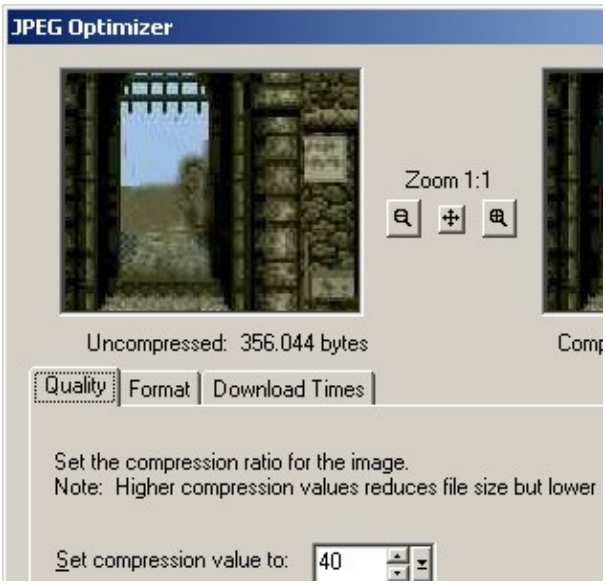
When collecting or preparing images for your game, you should make efforts to reduce image size and save bandwidth, while preserving the necessary image quality for your game to look good.

Tip: Modem users can get annoyed while waiting big images to load, so it is always a good idea to test play your game while using a standard dial-up modem. This way you can identify images which need to be reduced in size.

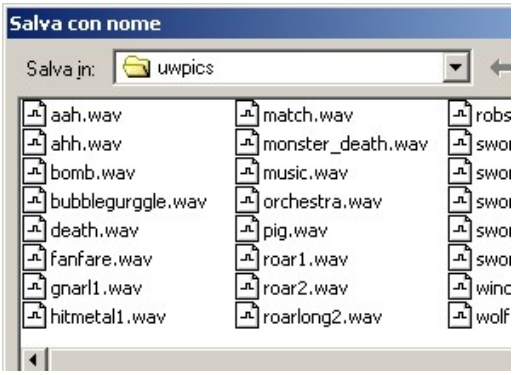
The following table, however, summarizes the current size recommendations for several file types.

Image type	**Typical size in pixels (WxH)**	**File type/ Extension**	**Typical size in Kbytes**
Scenes	400x300	JPG	10K to 20K
Icons	32x32	GIF (transp.)	1K
Item image	100x100	GIF (transp.)	5K to 10K
Character image	65x100	GIF (transp., animated)	5K to 15K
Icons to be used also as item images	48x48	GIF (transp.)	5K
Sound effects	-	WAV	5K-10K
Background music	-	MID	up to 70K

Tip: For images, you will greatly benefit from using an image manipulation software to properly compress the images to the desired size. Paint Shop Pro for example (download a free trial version from: http://www.jasc.com), has got some nice Export features (GIF optimizer, JPG optimizer, PNG optimizer) with which you can vary compression parameters and see the resulting file size, while assessing image quality:



Tip: Sounds effects can be found on http://www.findsounds.com and you can easily adapt them for your game by reducing their size with the Windows Sound Recorder, which gives you the possibility to change the sample's format. The smaller the quality, the smaller the file. Just click on the "**Change…**" button in the **Save As** dialog:



1.4.4 Developing multi-area games

Once your newly-created game becomes stable and begins to grow, you are tempted to make it bigger and bigger to attract new visitors.
The following rules will enable you to combine areas into a complex, composite game:

- Each area of your composite game is implemented as a DimensioneX WORLD. WORLD and area are treated as synonyms here.
- Each WORLD of the your composite game will belong to a same CLUSTER (see CLUSTER parameter in the WORLD definition, section 2.2). The big, composite game *is* a CLUSTER.
- Each WORLD of the same CLUSTER should share the same code for managing common objects. It is a good idea to use shareed code libraries that you **Include** (see the **Include** directive in section 3.3) in all areas of the same cluster.
- During the game, objects can be moved from an area to another of the same cluster by properly using the **MoveOutside** instruction. Please see section 3.11.14 - MoveOutside *what*, *areaid* for details.
- During normal game server operation, it is webmaster's responsability to ensure that all areas of the same cluster are loaded in memory. In future versions this will be ensured by the system automatically.
- Saved game profiles are stored in a common file, shared by all worlds belonging to the same cluster. The player may login back in whatever area within the cluster: he will be automatically brought to the area in which he/she saved the game.

1.4.5 The Administrator's page

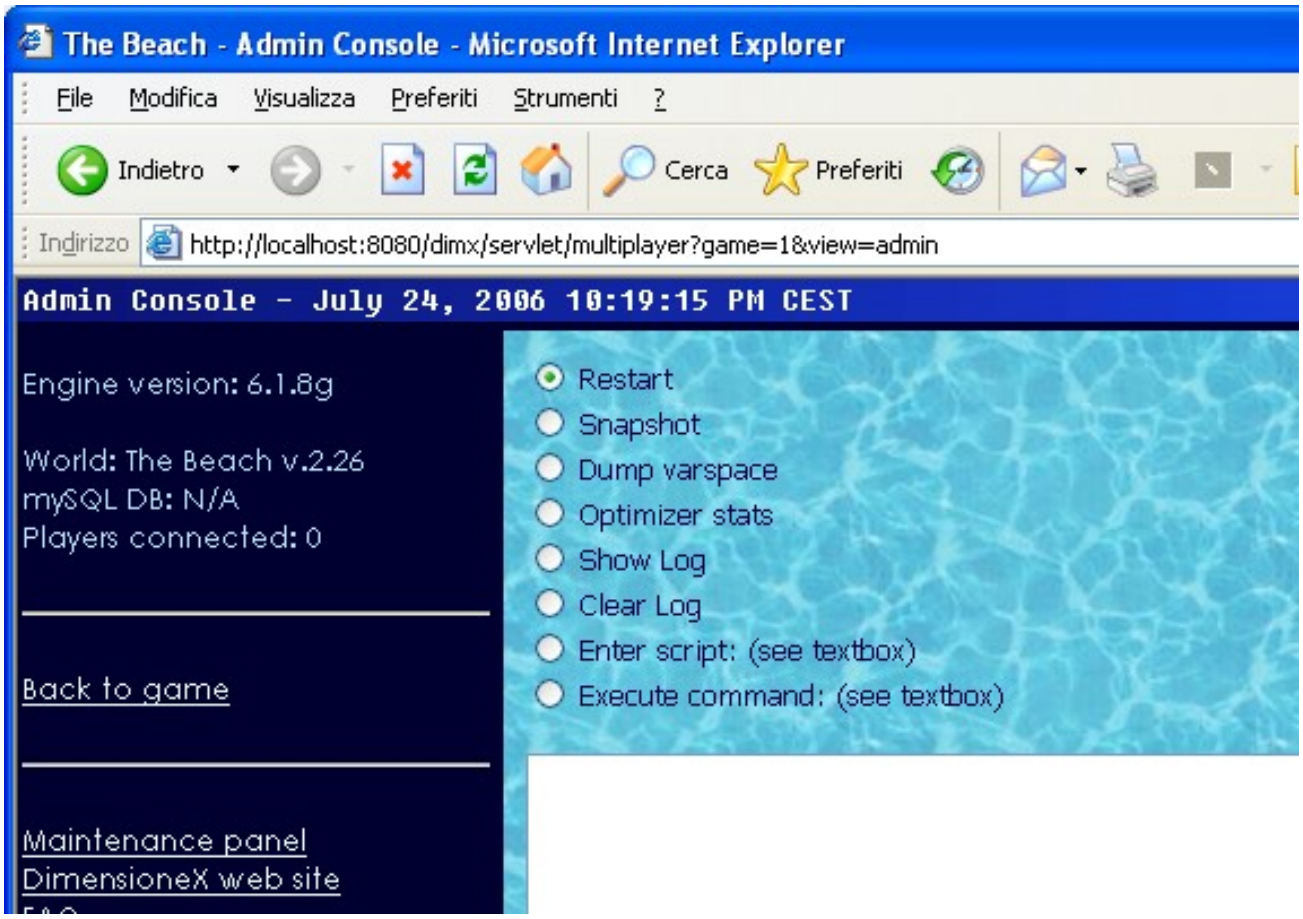
Most of the game developer's tasks can be carried out via the Administrator's page. This page is called by explicitly requesting the "admin" view to the DimensioneX game engine on the URL, like this:

http://*your-game-url-here*&view=admin

Example, for game in slot 1:

http://localhost:8080/dimx/servlet/multiplayer?game=1**&view=admin**

The Administration page will pop up:



Note: When calling the Administration page, if you omit the game=# specification the Admin page will refer to the game you have accessed most lately.

1.4.5.1 Functionalities

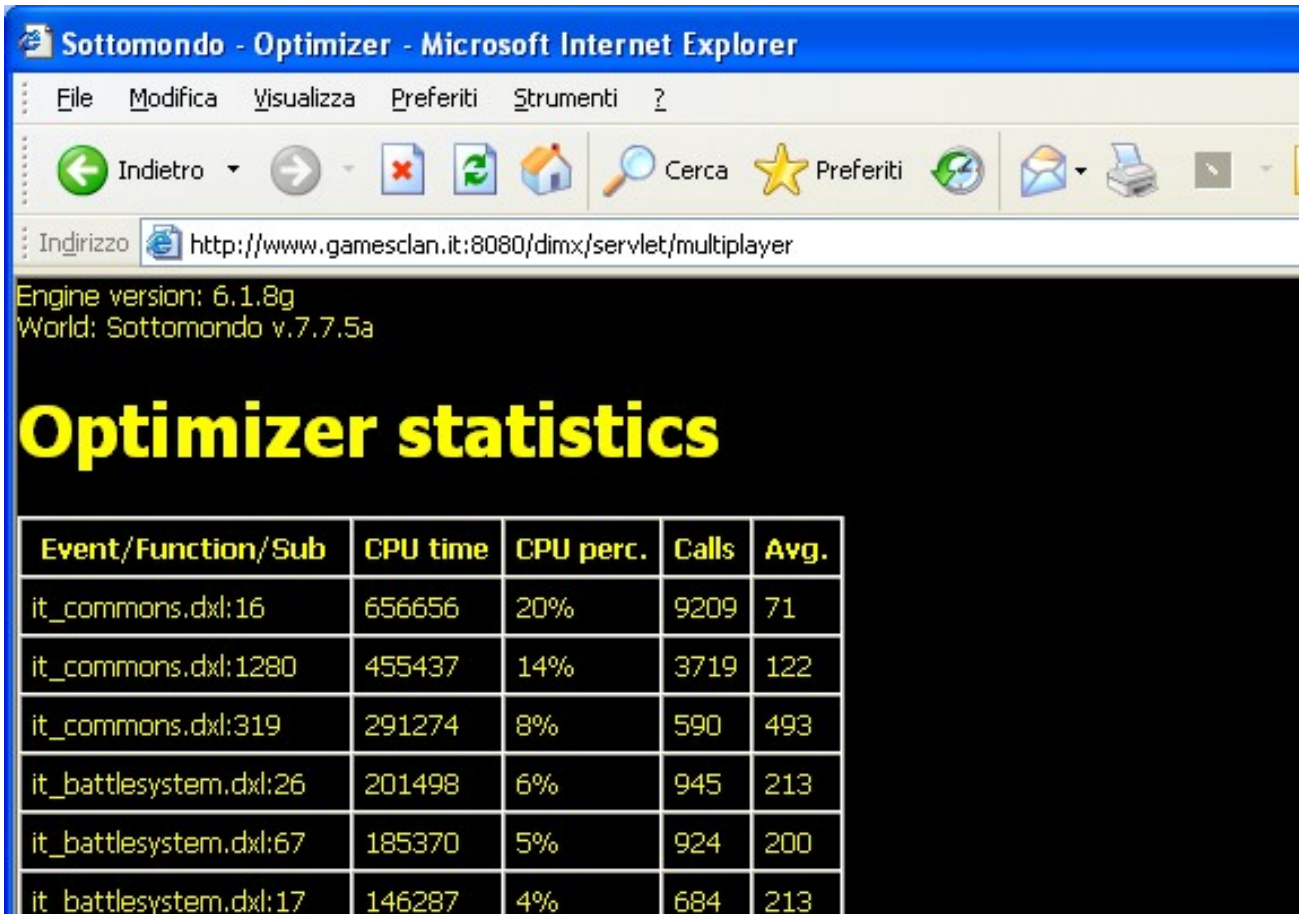
- Engine version, World name and version, Availability of DB connection, number of players connected. Displayed for your reference.
- **Restart** command. Select this one to re-loads settings and world file. All game situation is lost.
- **Snapshot** command. Select this one to get a snapshot of the game at the moment. Useful to inspect the current world, see where objects are, status of internal variables, etc.
- **Optimizer stats** command. Provides info about how your script code is actually performing. Number of calls, consumed CPU time, percentage, runtime errors. The highest names in the list might need optimisation.
- **Show log** command. Prints the contents of the current world's log. Even thoug you have set logging to 'none' it may contain useful about occurred errors (if any).
- **Clear log** command. Select this one to clear current world log.
- **Enter Script**. Type some Function/Sub/EVENT in the text area below, and use this command to add this code to the running game. If the Funcion Id matches an existing one, this one will be overwritten.
- **Execute command**. Type a valid command in the text area, and use this function to make the game engine to execute it. For example, to broadcast a message to all the players in the game:

```
Speak SYS,$WORLD,"NOTICE TO PLAYERS: The game will be restarted in 5 minutes"
```

- **Password** field. Type here your admin password (this one should have been set in the worldnav#.properties file)
- **Back to game** link. Reverts back to normal game view.
- **Maintenance** panel. Leads to the maintenance panel, which allows you to perform other admin tasks.
- **DimensioneX web site, FAQ, Documentation, Support** links. Quick links to programming aids.

1.4.5.2 Optimizer statistics

Provides info about how your script code is actually performing.
The page will look like this:



Event/Function/Sub	CPU time	CPU perc.	Calls	Avg.
it_commons.dxl:16	656656	20%	9209	71
it_commons.dxl:1280	455437	14%	3719	122
it_commons.dxl:319	291274	8%	590	493
it_battlesystem.dxl:26	201498	6%	945	213
it_battlesystem.dxl:67	185370	5%	924	200
it_battlesystem.dxl:17	146287	4%	684	213

Here is the meaning of the provided information:

- Engine version, World name and version. Displayed for your reference.
- **Event/Function/Sub** column. Your scripts, sorted from the most time-consuming to the less time-consuming. The software displays the file name where the script has been read, and the line number at which the script begins. The name of the function/Sub/Event is not displayed, to see it, open the specified file at the specified line.
- **CPU time** column. Consumed time, in milliseconds (1000 milliseconds = 1 second).

- **CPU percent** column. Percentage of the total execution time. This is another way to express the CPU time information, referred to the total consumed time (displayed at the bottom).
- **Calls** column. Number of times that the Sub/Function has been called.
- **Avg.** column. Average running time per each call, in milliseconds.
- **Total consumption:** Consumed time, total
- **Total calls:** Total number of calls
- **Runtime errors:** Number of runtime errors occurred since the start. All these errors have been logged on the debug#.log file.

If you look for good performance, since the list is sorted by relevance, items on top are the best candidate for inspection and optimisation.
Consider that inner calls to other subroutines do sum up in the caller routine consumed time. Take this into account and don't be surprised if the most time-consuming sub is one that just calls other scripts.

1.4.6 The Maintenance page

The maintenance page is called from the Administrator's page and from here you can perform other tasks at server level. You might need to know one or more admin passwords to perform the desired operations.
The page looks like this:

1.4.6.1 Functionalities

- **Game server instance** drop-down list. Decides on which world the chosen command will act on.
- **Admin password** field: Type here the administrator's password for the chosen command to work on the selected world/game. The password must be the one of the selected slot, OR the server's admin password.
- **Set up this game** command: Select this one and write a value in the text field to set up a game in the chosen slot. A correct value can either be a file name (if the DXW resides on the server's system folder) or a complete URL (if the DXW file resides elsewhere, such as an external web site)
- **Select game profiles** command: Selects and possibly erases saved games profiles from the game's database. This command only acts on the filesystem-based database (.SAV file). Just select the command and complete the remaining part of the line (all/date before…/version less than…), the command has a further confirmation screen.
- **Check DB Connection / Initialize**. Displays currently configured database driver and connection string being used. Connects to the DB, prints the status. If the database has not been initialized, it creates the necessary tables for their future use.
- **Game profiles – store SAV file into DB**. Copies all contents of the .SAV file into the mySQL database (valid connection needed)
- **Game profiles – export DB → SAV file**. Does the converse, that is, copies all contents of the mySQL database of game profiles (valid connection needed) into the text-based SAV file on the local filesystem.

No password is needed for the following functions:

- **Go to game's Admin console** command. Brings you back to the Admin command for the selected game.
- **Go to game** command. Brings you back to the selected game.

2 DimensioneX DXW Reference

At a first glance, a game for DimensioneX is described by a single .DXW file, coded according to a well-defined format. The structure and the syntax of the file is described by this and the following sections. DXW files can, however, be created from multiple DXW-formatted files by use of the Include statement. More on the Include statement below.

The game is also made complete by a set of multimedia files (pictures and sound files) that you will have to provide unless you want to develop a text-only multiplayer adventure.

2.1 Structure of the .DXW files

Each game environment is described in a DXW file, which is actually a text file formatted using a well defined syntax. In this section we describe the DXW file structure.

A file for DimensioneX, say "world.dxw" is more or less structured as follows:

```
WORLD
      ' World attributes go here

GUI
      ' Graphic-User-Interface settings and PANEL definitions go here
END_GUI

ROOMS
      ' ROOM definitions go here
END_ROOMS

LINKS
      ' LINK definitions go here
END_LINKS

CHARACTERS
      '  CHARACTER definitions go here
END_CHARACTERS

ITEMS
      '  ITEM / VEHICLE definitions go here
END_ITEMS

SETS
      '  SET definitions go here
END_SETS

SCRIPTS
      ' Your own scripts will go here
END_SCRIPTS

END_WORLD
```

- ✓ The file is structured in 7 sections: WORLD, GUI, ROOMS, LINKS, CHARACTERS, ITEMS, SETS, SCRIPTS. The first section (WORLD) contains the remaining seven.
- ✓ Each section has a closing keyword (END_GUI, END_ROOMS, END_LINKS, etc)
- ✓ Each section inside WORLD is optional
- ✓ Each section is described in detail in the following sections
- ✓ Text beginning with a quote (') is a comment and is ignored by the parser.

2.2 WORLD

The WORLD object represents the whole game environment.

Here is the syntax for its definition, with sample values. Square brackets indicate optional items. The attributes below can appear in any order.

```
WORLD
      NAME               world_name
      IMAGESFOLDER       http://www.dimensionex.net/pics/
      - or -
      IMAGESFOLDER_LOCAL      http://localhost:8080/pics/
      IMAGESFOLDER_PUBLIC     http://mywebsite.com/pics/

      [ ENCODING         ANSI ]
      [ VERSION          1.0 ]
      [ AUTHOR           John Smith ]
      [ AUTHOREMAIL      name@domain.net ]
      [ SITE             http://www.dimensionex.net/sitename ]
      [ HELP             http://www.dimensionex.net/en/quick_start.htm ]
      [ COUNTERHTML      html_code_here ]
      [ INTERPHONE       level ]
      [ SAVEGAME_PERSISTENCE  2 ]
      [ MUTING           1 ]
      [ CLUSTER          clusterid ]
      [ CUSTCMDPROC      custom.command.processor.classname ]

[Include "AnotherLibraryFile.dxl" ]

[GUI section]
[ROOMS section]
[LINKS section]
[CHARACTERS section]
[ITEMS section]
[SETS section]
[SCRIPTS section]

END_WORLD
```

Here you find a description of each world attribute. All attributes are optional, as defaults are defined for each one of them, but without specifying at least the IMAGESFOLDER attribute, your game will not look properly.

WORLD.Attribute	Meaning and values

NAME	Game title. Serves as a world ID.
IMAGESFOLDER	URL of the folder where most of the game's images and media files are contained. This setting applies whatever **serverType** setting you have in the worldnav##.properties file. **Important:** Do specify a final slash and do not use DOS-like backslashes ("\"). All the images used by your game are generally kept in one single folder which will be hosted on a web space. The most performing solution would be to separate the multimedia content web server and the game server, so CPU and bandwidth is not wasted for transferring images and allows quick management of the game. However, see the "documentation" section on the DimensioneX site to learn more about where to host multimedia content of your game (there's a knowledge base article on this topic).
IMAGESFOLDER_PUBLIC	URL of the folder where most of the game's images and media files are contained *when the game will be published to public*. This setting applies (and has priority over IMAGESFOLDER) only if you have set **serverType=public** in the worldnav##.properties file.
IMAGESFOLDER_LOCAL	URL of the folder where most of the game's images and media files are contained *when the game is being tested on local computer*. This setting applies (and has priority over IMAGESFOLDER) only if you have set **serverType=local** in the worldnav##.properties file.
ENCODING	(Optional) Specifies which encoding format has been used for this source file. This is important for rendering local (non-US) charsets correctly. Acepted values are: UTF-8 or ANSI The default value will be UTF-8.
VERSION	(Optional) Version of this game. Do not confuse it with the DimensioneX version, which is completely other thing.
AUTHOR	(Optional) Author's name for this game
AUTHOREMAIL	(Optional) Author's e-mail
SITE	(Optional) URL of the official game site
HELP	(Optional) URL of the file to be linked to the "Help" button. So, each game can have its own help system.
COUNTERHTML	(Optional) HTML string to allow the insertion of a counter. Get a free counter from a web service provider and then paste here the provided code so you can autonomously track usage of your game.
INTERPHONE	(Optional) level can be 0, 1 or 2 0 = speech is heard only within the same room 1 = (default) speech is heard word-wide but vehicle/item provide privacy 2 = speech is heard world-wide
SAVEGAME_PERSISTENCE	(Optional) 0 to 2. Specifies the persistence level of saved games. 0 = Saved game is canceled when game is restarted 1 = Saved game is canceled when game version changes 2 = Saved game is always valid (default)
CLUSTER	(Optional) specifies that the world belongs to a cluster with the specified ID.

	This is used to make WORLDs to act as areas of bigger, multi-area games. *clusterid* can be any valid identifier
MUTING	Specifies the level of the MUTING system. Available values are: 0 = No muting at all 1 = Basic – Each time a stopword is used in speech, the speech is censored, but the player is allowed to speak immediately after 2 = Normal (default) – The first time a stopword is used in speech, the player is muted for 1 minute, then allowed to speak again. 3 = Cumulative - Each time a stopword is used in speech, the player is muted for 1 minute. This penalizes repeated attempts to break the system
INCLUDE	Specify another DXW-formatted textfile to be Included or inserted as part of this world definition. These files are considered to be reusable "library" files and often use a .DXL extension. The filename must be enclosed in quotes and can be either an absolute URL (eg. http://www.myserver.com/folder/file.ext) or a relative path. In this case, the path is intended to be relative to the folder containing the main DXW source code. Includes can be used in ALL areas of the DXW file. Some examples: Include "MyWorld_Rooms.dxl" Include "MyWorld_Items.dxl" Include "MyWorld_Characters.dxl" Included files can contain other **Include** statements. Beware not to generate infinite include recursion!
CUSTCMDPROC	The game engine has certain Java hooks that allow Java developers to extend or bypass normal game processing. Interfaces available for such processing can generally be found in the org.dimx package of the distribution. Early examples of available hook points are the ISayProcessor for extending default handling of the SAY command and the IWorldInitProcessor allowing custom processors to include themselves in pre- and post- World initialization.

2.3 GUI section

The GUI section is optional and describes the command panels that will used in the game.
Inside it, each PANEL tag describes a command panel in the game. INCLUDE statements could be used to include libraries of reusable PANEL and PAGE definitions.

GUI section's syntax is as follows (square brackets indicate optional parts, things which are not written in bold can be changed by you):

```
GUI
[ SCENE SIZE    350x235]
[ SCENE LOOK    lookType ]
[ SCREEN SIZE   800x600]
[ LOGOSRC       gamelogo.jpg ]
[ MAP           gamemap.jpg ]
```

```
[ COMPASS      true ]
[ MSGLISTSIZE  3 ]
[ SKINS        skinspec, skinspec, … ]
[ SHOW PROPERTIES   property1, property2, … ]
[ HOOKS cmd1=event1,cmd2=event2, … ]

[PANEL definition]
[PANEL definition ...]

[PAGE definition]
[PAGE definition ...]

END_GUI
```

Tag	Meaning and values
SCENE SIZE	(Optional) Specifies the default size of the rooms' scenes, in pixels. Format: *width*x*height*. This is just a standard value, of course. You can design rooms with scenes of any size you like later. The default value is 350x235 which may be a good value for 640x480 screens, but rather small for bigger screens.
SCREEN SIZE	(Optional) Specifies the size of the "default" screen, in pixels. Format: *width*x*height*. The "default" screen we are talking about here is the screen for for which the game was designed (typically the one used by the game programmer dureing design and testing). The value for SCENE SIZE should enable a "perfect" display on this "default" screen. The game engine is able to automatically reproportion images so that the display is correct on different screen sizes. So, if a game with SCENE SIZE 350x200 was designed for a 640x480 screen, we simply state so by using the SCREEN SIZE tag. This way, when a player will be using a larger screen (say, 1024x768) the images will be proportionally enlarged.
SCENE LOOK	(Optional) Determines how the scene will look like in the standard view. Possible values for *lookType* are: `1STPERSON (default)` The scene is seen in 1st person (player is not displayed) `3RDPERSON` The scene is seen in 3rd person (player is displayed)
LOGOSRC	(Optional) URL of the game logo. If the URL is without the "http://" part, then it will be searched in the IMAGESFOLDER, as specified in the initial WORLD definition.
MAP	(Optional) URL of the game's default map. If the URL is without the "http://" part, then it will be searched in the IMAGESFOLDER. Map's dimensions should be 200x200
COMPASS	(Optional) "true" or "false". Specifies if the compass has to be displayed in the top bar. The default setting is **false**, which means that the compass is displayed only when it's necessary

	(eg. you are in a room with more than one picture).
MSGLISTSIZE	(Optional) Tells how many messages can be kept inside the player messages frame. The standard value is 3, but you may want to use a bigger value if you want to implement a chat site.
SKINS	(Optional) List of comma-separated skin (DXS) files specifiers (see next box below). They have to be listed in order of preference, the first will be used and proposed by default. If, instead of the skin file name the word "default" is specified, then the default skin is loaded. If the SKIN attribute is omitted, the default skin is used. Consult chapter 4 for the syntax of DXS files.
skinspec	The skin specifiers could be one of the following: ❑ `default` use the built-in default skin. This skin assumes that skins' graphic elements have default names and are located in the default IMAGEFOLDER ❑ `filename.dxs` loads the skin from the specified file. The file is searched in the current DimensoneX system folder (the same where the world description is located). A relative path may also be used. ❑ `http://www.mydomain.com/path/filename.dxs` will make the game engine to load the skin from the specified file via an HTTP connection. You just have to ensure the URL is reachable from the server at world's startup.
HOOKS	(Optional) Defines a number of server events that can be called from outside. See HOOKS definition below.
SHOW PROPERTIES	(Optional) Defines a set of properties to be displayed. By default, the game engine shows only those properties whose name begin with a capital letter.
PANEL	(Optional) Defines a custom panel which could be used in the game. See PANEL definition below.

2.4 HOOKS definition

HOOKS defines commands that the game server will accept from the outside, regardless of the state of the client (in-game player or not). Each command you specify in the HOOKS line will be associated to an event defined by you. Then, in the event code, it will be up to you what to process and what to output.

Syntax:

```
[ HOOKS hookID=eventName, … ]
```

- The function handling the event can have input parameters which will be specified by the user in the URL used for the call. The programmer may read the input parameters also via the **input()** set.
- Any output can be done via the **Print** instruction.

- The hook will be activated by opening the game's URL and using the **cmd** parameter with the specified hook ID. So you activate your event by opening an URL like this:
- http://localhost:8080/dimx/servlet/multiplayer?game=1&**cmd=hookID**&param1=foobar

2.4.1 Example

In the following example, we define a hook.

- We want to enable a way to peek into the game and quickly locate objects and characters, even when we are not connected.
- We want that, when the server receives the command: **findobj** on the game's URL, the event **doFindObj()** is triggered and, after getting the name of the object we are searching for, it outputs the results of the search in a plain text format.
- The name of the object we are searching for will be specified in the **obj** parameter.

```
GUI
      HOOKS       findobj=doFindObj   ' when you receive 'findobj' call doFindObj()
…

EVENTS
Function doFindObj(obj) 'we expect a parameter named obj
      'obj = input("obj")
      'Print "input: " + input + "<br>"
      'Print "obj: " + obj + "<br>"
      'Print "Searching: " + obj + "<br>"
      Dim c
      For Each c In getItemsIn($WORLD)
            If InStr(c.name,obj)
                  found=1
                  Print $AGENT,"<LI>" + c.name + " is in: " + c.container.name
            End_If
      Next
      For Each c In getCharactersIn($WORLD)
            If InStr(c.name,obj)
                  found=1
                  Print $AGENT,"<LI>" + c.name + " is in: " + c.container.name
            End_If
      Next
      If Not(found)
            Print "Not found: '" + obj + "'"
      End_If
End_Function
END_EVENTS
```

The command must be specified in the game's URL by using the **cmd** parameter. So you activate this hook by opening an URL like this:
http://localhost:8080/dimx/servlet/multiplayer?game=1&cmd=findobj&obj=spell

By opening the above URL you will get the result.

It should be now easy to understand that this can be used to integrate your game with any web based application.

Security notice: Be aware that when using HOOKS you open a breach into your game. While ordinary dimensioneX commands require that the user is connected or that he specifies the admin password, **any request calling your hook event will be accepted by the game server**. The task of checking who is performing the request, from what site/application, or how often is completely up to the programmer.

2.5 PANEL definition

Any PANEL definition must be placed in the GUI section and defines a controls panel to be used in the game.

GUI section's syntax is as follows (square brackets indicate optional parts):

```
PANEL panelId [VERSION OF parentPanelId]

[BUTTON definition | CR definition | MAP definition | TEXTBOX
definition | LABEL definition | DROPDOWN definition | DELETE spec ]
[more control definitions...]
```

The VERSION OF tag may be added to build a modified version of an existing panel, so that you just add additional controls, or delete existing ones.

Parameters	Meaning
panelId	Identifier of the panel (mandatory) If the panelId identifier represents an existing panel, the new definition will replace the existing one. **Note**: "*default*", "*chat*", "*connecting*" and "*exiting*" are IDs of pre-defined panels. "*default*" is the panel to be used by default at the game's startup and in controls' quick definitions. "*chat*" is an alternate, simplified set of commands which can be used for chatting purposes, by means of the **SetPanel** instruction (see section 3.11.25, in the SmallBasic reference). "*connecting*" is the panel to be used while connecting. "*exiting*" is the panel used when a "Save and Exit" command is used.
parentPanelId	(Optional) Identifier of an existing panel from which commands should be imported.

As it should be clear, a PANEL tag just specifies the PANEL's id for later reference, and acts as a container for the controls' specifications. For more information about them, see *2.5.2 - How PANELs work*.

2.5.1 Pre-defined PANELs

DimensioneX has got a number of pre-defined PANELS which, in turn, consist of pre-defined controls. Please remember that all defined PANELs can be inspected by using the DimensioneX **Administrator's** screen, selecting **Snapshot** and then **Panels**.

panel id	what is it?	built-in control IDs
Default	It is the panel used normally – all commands available.	go, look, use, use2, _sep4, open, close, pick, drop, _sep9, search, hide, put, give, _sep14, enter, exit, _sep17, _sep18, txtBox, say, _sep21, _sep22, clear, help, _sep25, save,savexit, logout
Chat	It is a simplified panel, useful for chatting and little else	go, look, _ctr2, _sep3, txtBox, _sep5, _sep6, use, _sep8, _sep9, help, savexit, logout
Connect	Initial connect screen	go, look, lbl0, username, _sep4, lbl1, skin, _sep7, lbl2, audio, _sep10, lbl3, music, _sep13, lbl4, screensize, _sep16, _sep17, help, login, lbl5, lbl6
Connecting	It is used by the system when connecting with a saved profile	go, look, lbl0, _sep3, pass, _sep5, _sep6, login, _sep8, _sep9, startscratch, cmd, _sep12, _sep13, return, _sep15, _sep16, help
Exiting	It is used by the system when doing "Save and Exit"	go, look, lbl1, txtBox, _sep4, _sep5, savexit, _sep7, return, _sep9, help

Note:
- _sep*** in the above table are separators (line breaks).
- Labels for the built-in controls of the pre-defined PANELs are directly taken from the msgs_xxx.properties language file.
- pre-defined controls (look, open, etc…) will trigger pre-defined EVENTs. Currently there's no documentation on this. So, to know exactly which EVENT is triggered by each command, you should simply use the command and then look on the **debug.log** file to see which EVENT is triggered (loof for "world.fireEvent").

2.5.2 How PANELs work

(a) Each game (or WORLD) at startup is initialized so that it uses the *default* PANEL.
(b) Each connecting player may have a player-scpecific commands panel. However, this player-specific commands panel is not set by default.

When a player is in the game, the controls he or she sees are determined by the following:

- player-specific commands (optional, as for point ***b*** above)

plus (+)

- room-specific commands (optional). If these are missing, then the commands from the world's current PANEL is used (normally the default one as specified in point ***a*** above).

The world's current panel can be changed via the SetPanel instruction (see SmallBasic guide, below)

Also the player's specific or room's specific panels can also be changed via the setPanel instruction. The combination of specific, and WORLD-wide panels gives the game programmer a great flexibility and power.

2.5.3 BUTTON definition

BUTTON defines a command button inside a PANEL definition.
The syntax depends on what button you need, as follows:

Syntax #1, for standard buttons:

```
BUTTON id,"Label","ToolTipText", Event [,EventModel]
```

Syntax #2, for ICON-enhanced buttons:

```
BUTTON id,"BalloonText","ToolTipText", Event [,EventModel]
     ICON src
```

Syntax #3, for IMAGE buttons:

```
BUTTON id,"BalloonText","ToolTipText", Event [,EventModel]
     IMAGE [widthxheight] src
```

Syntax #4, for quick import from *default* panel

```
BUTTON id
     [ICON src]
```

Syntax #1 is the standard one.

Syntax #2 is a the recommended one. You specify an icon to be shown starting from the upper-left corner of the button. The icon is typically a 16x16 transparent GIF image (but you can actually use imges of any size and type depending on the effect you want to achieve). This syntax will cope with multiple languages and colours.

Syntax #3 adds a line for specifying the buttons' IMAGE. Consult 2.10 - IMAGE at page 48 for more information on the IMAGE definition.

Syntax #4 is also said "quick definition". It is used to quickly reference a standard, existing BUTTON control which is present in the "default" commands PANEL without having to re-define it. The commands look, by example, must be included in all panels so this syntax can be used to quickly do the job. Also the command logout, savexit, help may be useful to include.
You can also specify an ICON tag so that an icon is added to standard commands.

Parameters	Meaning
Id	Identifier of the button (mandatory) **Note**: The following Ids have a special meaning and are automatically dealt with by the DimensioneX Javascript client. Don't use these ones if you would like to implement different, custom commands: **look, go, logout, savexit, return, rotate, use, use2, drop, pick, put, give,**

	say, search, hide, clear, rotr, rotl You can use the above, however, to change the look of the buttons but you won't be able to change the command's standard behaviour.
Label	Text of the button as it will appear on the command panel. Double quotes recommended.
BalloonText	Text which will appear in the yellow balloon hints when pausing with the mouse on an IMAGE button's surface. Same as for *Label*.
ToolTipText	Tip text for the toolbar hints. Double quotes are mandatory. The triple dots are automatically handled by the client and substituted with actual object names. A typical Tip Text are "Attack …" "Trade … with …" "Give … with …" and so on.
Event	ID of for the event or full function call to be triggered when the command is used. If you specify parameters separated by commas, you MUST place the code between double quotes.
EventModel	Describes what parameters the command uses and how they are mapped into the owner/target variables according to the DimensioneX avent model. Values can be one of the following: (none) : no parameters O : 1 object parameter (owner) OO : 2 object parameters (owner, target) T : 1 text parameter (target) TO : 1 text parameter, 1 object parameter (target, owner)

Here is an example, which defines a control which rolls a dice, and includes the code for the associated EVENT to exploit this new custom command:

```
GUI
PANEL default
      BUTTON create, "Create Object", "Create Object!", onCreate
      BUTTON destroy, "Destroy Object", "Destroy Object!", onDestroy
      BUTTON rolldice, "Roll Dice", "Roll Dice!", onRollDice
      BUTTON look
      BUTTON logout
END_GUI

...
SCRIPTS

EVENT onRollDice
      nr = rndInt(6)
      Speak SYS,$WORLD,$AGENT.name + " has rolled the dice and the result is " + nr

END_SCRIPTS
```

Another example, showing how a full function call can be used in Event. Here, the doAffinity sub is used to handle all four buttons, in fact the working logic is depending on the value of the four parameters.

```
PANEL paffinity
      BUTTON waterplus, "+", "Command", "doAffinity(1,0,0,0)"
      BUTTON water, "Water", "Water", helpAffi
            IMAGE 32x32       elem1.gif
      LABEL " Water"
      BUTTON waterminus, "-", "Command", "doAffinity(-1,0,0,0)"
      LABEL "<BR />"
      BUTTON earthplus, "+", "Command", "doAffinity(0,1,0,0)"
      BUTTON earth, "Earth", "Earth", helpAffi
            IMAGE 32x32       elem2.gif
      LABEL " Earth"
      BUTTON earthminus, "-", "Command", "doAffinity(0,-1,0,0)"
      LABEL "<BR />"
      BUTTON windplus, "+", "Command", "doAffinity(0,0,1,0)"
      BUTTON wind, "Wind", "Wind", helpAffi
            IMAGE 32x32       elem3.gif
      LABEL " Wind"
      BUTTON windminus, "-", "Command", "doAffinity(0,0,-1,0)"
      LABEL "<BR />"
      BUTTON fireplus, "+", "Command", "doAffinity(0,0,0,1)"
      BUTTON fire, "Fire", "Fire", helpAffi
            IMAGE 32x32       elem4.gif
      LABEL " Fire"
      BUTTON fireminus, "-", "Command", "doAffinity(0,0,0,-1)"
      LABEL "<BR /><BR /><INPUT TYPE=hidden NAME=affi VALUE=1>"
      BUTTON dothis,"Done","Command", doCommand

Sub doAffinity(e1,e2,e3,e4)
      'Event code...
End_Sub
```

2.5.4 CR definition

CR defines a carriage return inside a PANEL definition. The syntax is simply as follows:

```
CR
```

2.5.5 MAP definition

MAP defines a map box inside a PANEL definition. The syntax is simply as follows:

```
MAP
```

The map image to be displayed must be defined in the GUI section by means of the MAP tag and can be changed according to any room by using the room's **map** attribute.

2.5.6 TEXTBOX definition

TEXTBOX defines a text box inside a PANEL definition. The syntax is as follows:

```
TEXTBOX [id]
```

The *id* is optional and will be returned as a key in the **input( )** set. If omitted, ir defaults to "txtBox"

2.5.7 LABEL definition

LABEL defines a label inside a PANEL definition. The syntax is simply as follows:

```
LABEL "free text"
```

Note: HTML code can be inserted inside double quotes. This element is indeed very powerful as it can be used to create very special panels, provided you have understood how the Javascript in the player's client works.

2.5.8 DROPDOWN definition

DROPDOWN defines a drop-down list inside a PANEL definition. The syntax is:

```
DROPDOWN id,Set,Default [,WorkingModel]
```

Parameters	Meaning
Id	Identifier of the control (mandatory) Avoid conflicts with reserved (and other custom) control IDs
Set	Expression to be evaluated at run-time which must return a SET containing identifiers and decriptions or an ARRAY containing decriptions to be used for building the dropdown list. You can use $OWNER to reference the player. For complex expressions you must enclose the expression between quotes, otherwise you will get an error. Example: "$OWNER.setChoices" is a valid expression, $OWNER.setChoices will report that there is an unexpected dot.
Default	Expression to be evaluated at run-time which should (theoretically) be one of the Ids of the *Set.* If so, the corresponding element will be pre-selected. Note: If *Set* will result in an ARRAY then this must be a zero-padded number of four digits. **Example** : 0002 sets the second element of the array as the default. You can use $OWNER to reference the player. For complex expressions you must enclose the expression between quotes, otherwise you will get an error.
WorkingModel	(Optional) Sets the working mode. Possible values: AUTOSUBMIT Note: In this release this setting will not produce any result. This functionality will be implemented in future releases.

2.5.9 DELETE specification

DELETE removes an existing control from the panel:
The syntax is:

```
DELETE id
```

Parameters	Meaning
Id	Identifier of the control to be removed (mandatory)

In the following example, we modify the connect panel so that it fits a horizontal frame:

```
PANEL connect VERSION OF connect
      CR
      DELETE _sep3
      DELETE _sep4
```

2.6 PAGE definition (formerly VIEW)

PAGE defines a custom HTML template which can be sent to the player's client.
Any PAGE definition must be placed in the GUI section according to the following syntax:

(square brackets indicate optional parts):

```
PAGE viewId , template
```

In practice, the specified template is loaded at startup time and then used at the programmer's need.

The template is actually an HTML file which you build with any suitable editor. When designing the view, you should insert in the HTML code the placeholders: **$scene, $icons, $inventory, $navpad, $messages** which will be replaced at run-time by the actual content.

Parameters	Meaning
viewId	Identifier of the view (mandatory) **Note**: "*default*" refers to the pre-defined view.
template	It is normally a .HTM file which resides in the media folder. You can specify a file name, a relative path, or an absolute URL starting with http:// This is also called HTML template. See next section to learn how to produce one.

The defined PAGEs can be used later by means of the **UseView** script instruction (See section 3.11.27).

2.6.1 HTML templates

HTML Templates are HTML files which can be used by the game engine to produce game views. They must be declared by using the PAGE tag in the GUI section (see above), and then called at programmer's need by using the script's **SendPage** instruction.

HTML templates can be prepared by the programmer by using any HTML editor (or any text editor if you like). They can include special placeholders which will be automatically replaced by game's real

values during game display. Also, the player's chosen skin is applied to the HTML template automatically.

The following is a list of currently supported placeholders:

Placeholder	Will be replaced with…
$display	Player's messages (avatar's thoughts)
$icons	Icons of objects in the current room.
$navpad	Navigation pad
$inventory	Player's inventory
$banner	ONLY the banner of focusized object normally seen at the top of a $scene. Typically used when NOT displaying the $scene.
$properties	Player's properties (all those beginning with a capital letter)
$scene	Scene display (usually the room's image), with focusized object on top. Note: this will always appear in the same position, that is at coordinates 1,21 of the screen. This will be fixed in later releases.

HTML templates can use media files (e.g. images) contained in your game's IMAGEDIR or anywhere else on the web by using an absolute URL starting with http://.

Tip: It is recommended that you compose any HTML template in your images folder, so you can peview the result in your browser. Then, when it looks OK, just move it to folder where you keep the game's DXW file.

The DimensioneX game engine expects to find the HTML templates in the same folder where the main game source resides.

2.7 ROOMS section

The ROOMS sections describes the rooms of the game. It simply contains one or more ROOM definitions. INCLUDE statements could be used to include reusable libraries of ROOM definitions.

2.8 ROOM

Each ROOM definition describes a room in the game.

Syntax: (square brackets indicate optional parts):

```
ROOM id  [DEFAULT]
     NAME Name goes here
     [DESCRIPTION Description]
     [ATTRLIST attrList]
     [PANEL panelId]
     [IMAGE definitions here … ]
```

The parameters are specified below. Don't include double quotes (") for specifying the values:

ROOM parameters	Meaning
Id	Identifier of the room (mandatory) **Basic principle**: ID, like all identifiers used in DimensioneX, can be made of letters and numbers, no spaces nor other characters. ID is a unique identier and cannot be used for other objects.
DEFAULT	(Optional) If this keyword is specified after the ROOM's ID, this room is used as a starting point when a player connects.
Name	Room's name (as displayed on the player's screen)
Description	(Optional) Description of the room. This is the text the user will read when entering in or looking at the room. **Dynamic descriptions** A description starting with the ampersand character, e.g. @myFunction($AGENT) Will produce a dynamic description by executing and using the return value of myFunction($AGENT).
PanelId	(Optional) Id of a control panel to be used in the room. If omitted, the world's default panel is used.
attrList	Comma-separated list of pairs, of the form: Attribute1=val1,Attribute2=val2 See also: 2.11 - Attributes/properties, ATTRLIST definitions - page 50. Note: If you would like the player's position to be tracked on the game's map you have to specify the *mapx* and *mapy* attributes. Example: ATTRLIST mapx=100,mapy=200
IMAGE	The ROOM definition may contain one or more IMAGE definitions which specify the images associated to the room itself. For more on this, see below

Note: When using transparent scene images (GIF and PNG formats), in order to set night/day sky color, use the WORLD's **bgcolor** property. Example:

```
Bgcolor = "#003355" ' Sets dark blue sky (night)
```

2.9 SHOW types

The SHOW tag is used in definitions of ITEMS and CHARACTERS to define how they should appear on the game screen. Here is the SHOW tag syntax:

```
SHOW showMode [showX,showY [FOR showFor] ]
```

2.9.1 showMode

showMode can take one of the following values:

OFFSCREEN	Specifies that the object is shown as an icon in the icons panel, outside the scene picture. This is the default behaviour for ITEMS. If an IMAGE is specified for the object, it can be seen when the player looks at the object. Note: (corresponds to object.showmode=0)
ONSCREEN	Specifies that the object is shown by its IMAGE on the scene picture. This is the default behaviour for CHARACTERS. Note: (corresponds to object.showmode=1)
ICON	Specifies that the object is shown as an icon on the scene picture. Note: (corresponds to object.showmode=2)

2.9.2 showX, showY

A pair of co-ordinates can be also specified, indicating the position in the scene picture where the lower-left corner of the image/icon should appear.

Notes

- The lower-left corner of the picture is co-ordinates (0,0).
- If unspecified, the position is automatically determined at random and by considering the scene pictures' SHOWAREA.
- If co-ordinates are not valid, they are automatically ignored and replaced by a valid random value.
- Currently, LINKS objects always show OFFSCREEN so if you would like to change their on-screen appearence (an open/closed door for instance), you should change programmatically the scene's picture.

2.9.3 showFor

If a FOR tag is specified, the object is shown using the specified co-ordinates only if the scene image's URL matches the provided one. In other words, *showFor* is an URL that locks the object's image to a specific scene image.

2.10 IMAGE

The IMAGE definition is widely used in the DimensioneX file syntax in order to describe an image being used.

An IMAGE is defined in one line according to the following syntax:

```
IMAGE [orientation] [dimensions] [ SHOWAREA [x1],[x2],y ] url
```

Example:

```
IMAGE N 400x300   SHOWAREA 100,300,10     start.jpg
```

The parameters are specified below. Don't include double quotes (") for specifying the values:

IMAGE attribute	Meaning

url	Defines the file representing the image.
	It can be either a fully-qualified, absolute URL or a relative URL (a simple file name is considered as such).
	
	✓ Absolute URLs are not recommended.
	✓ URLs beginning with a slash are considered relative to the server hosting the game.
	For example, the following:
	`/pics/scroll.gif`
	always refers to the /pics/ folder in the server hosting the game.
	
	✓ Relative URLs not starting with a slash are considered relative to the IMAGESFOLDER attribute which has been defined for this WORLD. This means that, by deafult, all images are assumed to be stored in the same IMAGESFOLDER.
	
	For example, the following:
	`scroll.gif`
	will be automatically modified so that the image will be fetched from the WORLD's IMAGESFOLDER.

dimensions	(Optional) Specifies width and height of the image in the form: 123x123. The first number is width, the second is height. This data are optional but, if specified, the effect as perceived by the user is that the page loads more quickly.
orientation	(Optional) Specifies the orientation which an observer should have in order to see that image. Possible values are N S E W U D for North South East West Up Down. Use this attribute in multiple IMAGE definitions in order to create navigable 360° views of the object.
SHOWAREA x1,x2,y	(Optional). Specifies the baseline along which any on-screen object or icon can be shown. The purpose of this information is to avoid erroneously displayed situations such as: objects flying in the air, characters standing on the water's surface, etc.

	The default values are: x1 = left side of the picture x2 = right side of the picture y = 10% of the picture's height Please note that, any invalid value is automatically replaced by its default value specified above. For this reason, if you would like not to specify a parameter, just use the conventional value –1, which the game engine will interpret as: "use the default".

Note: The IMAGE tag is **not** used for specifying icons, which are dealt with by a specific ICON attribute.

See also *3.8.8 - Object model for: IMAGE* for information about how to handle IMAGE objects in the scripts.

2.10.1 Allowed formats

Allowed formats are: JPG (Jpeg), GIF, PNG for images. Please be sure to read carefully section *1.4.3 - About game images and sounds* for useful tips about image/clip optimization.

In the IMAGE tag you can use also:

- **Macromedia Flash (.SWF)**
  animations in .SWF format. Please note that it is not possile to see objects placed on top of embedded objects, such as SWF animations.
- **QuickTime (.MOV)**
  panoramic images or 360° images coded in the QuickTime VR Format, QuickTime video clips. Please note that it is not possile to see objects placed on top of embedded objects, such as MOV clips.
- More image formats may work, depending on the user's browser.

2.11 Attributes/properties, ATTRLIST definitions

This definition is relevant to all objects.

The additional ***attributes*** help you to better qualify and specify the behaviour of an object as they are treated like *local variables* (or *properties*) of that object.
By means of an additional attribute you can specify for instance whether an item can be picked or not ("pickable"), if it is visible ("hidden"), if it is "open", and so on.
A number of pre-defined attributes are ready to be used. Plus, the developer can define his own attributes which can help him to control the game plot by means of *events*.
Here in this section you find a list of pre-defined, standard attributes which are automatically recognised and managed by the system to handle the most common situations.

Basic principle: Attributes are specified as comma-separated list of "*key*=*value*" pairs. The part "=value" can be omitted, in that case =true is considered the default value.

Attribute/property name	Meaning for the game
`Open`	Tells whether an object is open or not. WAY objects not having this attribute are implicitely open.
`openable`	Tells whether an object can be opened by a player. If it has open or locked attributes, the object is implicitly openable. You should use this one in the form openable=0 to specify that cannot be opened, and this does not depend from a lock. For example, a sealed metal box cannot be opened, and it has no lock to open with whatever key. So the message the player will get when trying to open it will be different.
`Locked`	Tells whether an object can be opened with a key. **Nota**: To programmatically open a "locked" object, you set the "open" attribute to 1 or true.
`pickable`	If an object can be picked or not. Implicitely is not.
`Hidden`	If an object is hidden or not. Implicitely is not.
`hideable`	If an item is hideable or not. Implicitely it is. You should use this one in the form hideable=0 to specify that cannot be hidden. Characters, ways and scenes cannot be hidden by the player.

Examples on the use of attributes can be found in the sample games provided with DimensioneX.
Note: the VOLUME and CAPACITY attributes are currently controlled by specific tags. See the section *2.11.2 - CAPACITY and VOLUME attributes*.

2.11.1 Hidden properties

Normally, player's and each examined object's properties are not displayed on the game screen. To show them, the programmer has got two options:

1) Use property names beginning with a CAPITAL letter. By default, all properties beginning with a capital letter are shown.

   Example: *Health* will be displayed, *health* will not.

2) Define a custom list of properties to be shown with the SHOW PROPERTIES directive in the GUI section. See *2.3 - GUI section* for more details.

2.11.2 CAPACITY and VOLUME attributes

Each object has a specific size or, better saaid, *volume* (this is to add realism, otherwise you may pick up inifinite objects) and, if an item is a container, it also has a *capacity*. Volume and capacity are measured in units, and there are default values for all objects, see the following table.

object type	default CAPACITY	default VOLUME
CHARACTER	5	10
ITEM	5	1
VEHICLE	10	10

In simple words, by default a player can carry 5 items of size 1, or an item of size 3 plus an of size 2, or… you got it.. You may want, however, create volume-less object by setting their volume to zero, or you may want to create huge containers with capacity 1000, 5000, 10000 …

2.12 LINKS, LINK

The LINKS section describes how the rooms are interconnected. This section contains any number of LINK or MLINK tags. INCLUDE statements could be used to access libraries of reusable LINK definitions.

Each LINK tag describes a bi-directional connection, (mono-directional if the MLINK keyword is used). Usually, doors between rooms are described as bi-directional LINKs.

LINK syntax (square brackets indicate optional parts):

```
LINK id from-to     [orientation]
     NAME Name goes here
     [DESCRIPTION textual description]
     [ATTRLIST attributes list]
     [ICON icon url]
     [SHOW show_type]
     [IMAGE definitions here … ]
```

MLINK works exactly the same, just replace LINK with MLINK and the link will be mono-directional.

The parameters are specified below. Don't include double quotes (") for specifying the values:

(M)LINK parameter	Meaning
Id	Passage identifier (mandatory)
Name	Passage name (this text appears next to passage icon) **Special value:** Specify "*" to let DimensioneX automatically use the ***orientation*** name (North /South and so on)
From	ID of the room from which the passage starts (mandatory)
To	ID of the room to which the passage leads (mandatory)

orientation	(Optional) Char specifying the direction, determines also the icon. Possible values are **N S W E U D** (for North South West East Up Down).
ICON	(Optional) Icon associated to this passage. If missing, the icon is determined by the *orientation*. If *orientation* parameter is also missing, then the default passage icon is used.
ATTRLIST	Comma-separated list of pairs, of the form: Attribute1=val1,Attribute2=val2 Use this attribute to specify features of this passage. For example: `open=0, locked` means closed and locked. See also: 2.11 - Attributes/properties, ATTRLIST definitions - page 50.
NAME	The name of the link, as it will be displayed
DESCRIPTION	Textual description.
SHOW	(Optional) defines the SHOW type for this link, default is OFFSCREEN. See *2.9 - SHOW types* for more details. Note: Can be specified on MLINKs only

LINK example:

```
LINK door1 P-End    W
     NAME door
     ATTRLIST  open=0,locked
     ICON icoDoor.gif
```

2.13 CHARACTERS section

This section describes the actors populating your game. This section can contain zero or more CHARACTER definitions. INCLUDE statements could be used to include libraries of reusable CHARACTER definitions.

.

CHARACTERS syntax (square brackets indicate optional parts):

```
CHARACTERS
[ATTRLIST attributes list]

[CHARACTER definition here…]
[CHARACTER definition here…]

END_CHARACTERS
```

The parameters are specified below. Don't include double quotes (") for specifying the values:

Parameter	Meaning
ATTRLIST	Comma-separated list of pairs, of the form: Attribute1=val1,Attribute2=val2 which specify the attributes that ALL the characters (especially: players) will have, and also their starting value. Use this attribute to implement useful parameters for the game like skill, strength, intelligence, etc. See also: 2.11 - Attributes/properties, ATTRLIST definitions - page 50.

example:

```
CHARACTERS
ATTRLIST Health=10,Strength=1,Dexterity=1

CHARACTER asia
      NAME        Asia
      DESCRIPTION Asia is the princess of the Reign.
      POSITION    End
      IMAGE             120x130     sm/chrAsia.gif

END_CHARACTER
```

2.14 CHARACTER

This tag defines a computer-controlled character.

CHARACTER syntax (square brackets indicate optional parts):

```
CHARACTER id [DEFAULT]
      NAME Name goes here
      [DESCRIPTION textual description]
      [ATTRLIST attributes list]
      [ACCEPTS list of object_ids]
      [POSITION room_id]
      [ICON icon url]
      [VOLUME int_value]
      [CAPACITY int_value]
      [SHOW type]
      [IMAGE definitions here … ]
```

Parameter	Meaning
ID	Identifier (mandatory)
NAME	Character's full name (as seen by the players)
DESCRIPTION	Description text which appears when you look at him
POSITION	(Optional) ID of the container (typically a ROOM object) representing its initial position. If not specified, the character

	will not be initially visible.
ATTRLIST	(Optional) Additional attributes list. It's a comma-separated list. These will be added to the default attributes, as specified in the CHARACTERS section. See also: 2.11 - Attributes/properties, ATTRLIST definitions - page 50.
ACCEPTS	(Optional) Comma-separated list of ITEM Ids and/or TYPE Ids of ITEMs that the CHARACTER would accept. To specify that **any object** is accapted, use the wildcard character: * Please note that, while you can give any item to any other player, computer-controlled characters accept nothing by default. To make the character to accept something, you have to tell so explicitly using this attribute.
DEFAULT	(Optional) If "true", this entity becomes the "system default", which is used to make system announcements such as "The game is over". In this case, it is recommended not to use the POSITION attribute.
ICON	(Optional) Icon of the actor. If not specified, the default one is used.
CAPACITY	(Optional) Has got sense only for those objects capable of cointaining other objects. If not specified, defaults to 5 units. See also *2.11.2 - CAPACITY and VOLUME attributes*.
VOLUME	(Optional) Represents the size of the object, expressed in *units*. Defaults to 1. You may want to give bigger volumes to bigger objects. See also *2.11.2 - CAPACITY and VOLUME attributes*.
SHOW	(Optional) defines the SHOW type for this character, default is ONSCREEN. See section on SHOW types for more details.
IMAGE	(Optional) Image(s) for the actor.

Example:

```
CHARACTER nkono
      NAME        N'Kono
      DESCRIPTION N'Kono is a Zulu warrior. What is he doing here?
      POSITION    R4
      IMAGE       80x200      sm/chrNkono.gif
```

2.15 ITEMS section

This section describes the items contained in our game. It may contain any number of ITEM and VEHICLE definitions, in any order. INCLUDE statements could be used to include libraries of reusable ITEM or VEHICLE definitions.

Syntax: (square brackets indicate optional parts):

```
ITEMS
      [ITEM definition]
      [VEHICLE definition]
      [... other ITEM or VEHICLE ...]
END_ITEMS
```

2.16 ITEM

This section describes an ITEM of the game.
Items are objects that do not represent people, but can be small objects, containers, even animals.

ITEM syntax (square brackets indicate optional parts):

```
ITEM id
     NAME Name goes here
     [DESCRIPTION textual description]
     [TYPE typeId]
     [ATTRLIST attributes list]
     [CAPACITY int_value]
     [VOLUME int_value]
     [POSITION room_id]
     [ICON icon url]
     [PANEL panelId]
     [SHOW type]
     [IMAGE definitions here … ]
     [INNER IMAGE …]
     [ZOOMIMAGE … ]
```

The parameters are specified below. Don't include double quotes (") for specifying the values:

Tag	Meaning
ID	Identifier (mandatory)
NAME	Name (as seen by the player)
DESCRIPTION	Descriptive text as seen when the player looks at
POSITION	Initial position ID. May be either another item or a character, or a room.
ATTRLIST	(Optional) Additional attributes list. It's a comma-separated list, see the related paragraph below.
TYPE	(Optional) Specifies a type identifier. The type identifier

	specifies the class this ITEM belongs to.
ICON	(Optional) Icon. If not specified, the default one as defined in the players' skin will be used.
CAPACITY	(Optional) Has got sense only for those objects capable of cointaining other objects. If not specified, defaults to 5 units. See also *2.11.2 - CAPACITY and VOLUME attributes*.
VOLUME	(Optional) Represents the size of the object, expressed in *units*. Defaults to 1. You may want to give bigger volumes to bigger objects. See also *2.11.2 - CAPACITY and VOLUME attributes*.
SHOW	(Optional) Specifies a SHOW type. OFFSCREEN is the default. See section about SHOW types for details.
IMAGE	(Optional) one or more IMAGE definitions which specify the images associated to the room itself. Definition of such tag is given in section *2.10 - IMAGE*
PANEL *PanelId*	(Optional) Specifies a control panel to be used inside the item (mostly used in VEHICLEs). If omitted, the world's default panel is used.
INNER IMAGE	(Optional) Definition of an IMAGE of the ITEM internal. This is used for big ITEMs which can contain players (typically VEHICLEs). Don't forget to specify INER IMAGE's width and height.
ZOOMIMAGE	(Optional) Definition of an IMAGE for the zoomed ITEM. This is used when the is looked at. Corresponds to the zoomImage property and it is never used for scene rendering, but just for zoom view.

Example:
a simple item

```
ITEM apple
      NAME        an apple
      ICON        icoMela.gif
      POSITION    beach
      DESCRIPTION A nice red apple... maybe it's good and tasty...
      ATTRLIST    pickable
      IMAGE       121x88      mela.gif
      SHOW            ICON
```

This example refers to an hamburger, owned by the "bodybuilder" character, it is pickable (but the body builder should agree!)

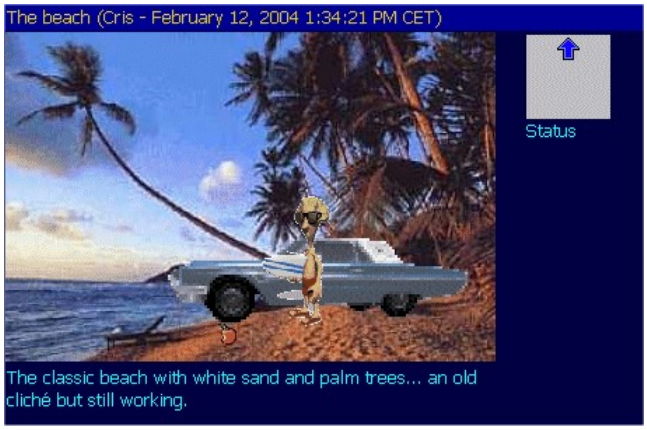
Notes:
For vehicles that can be driven please use the VEHICLE type, see below.
Several identical items will require identical tags with different IDs.

2.17 VEHICLE

This section describes a VEHICLE of the game.
Vehicles are identical to objects except for the following:

- Inner images can have transparent areas, and through these areas the player will see the room in which the vehicle is.
- When in a vehicle, the navigational controls will appear and they will be identical to those of the ROOM in which the vehicle is in. By using navigational controls, the player can actually drive the vehicle in the game

The screenshots below will illustrate the effect of the car example included at the end of this section.



Picture 1 - Before entering the VEHICLE

Picture 2 - After entering a VEHICLE

Syntax is identical to that of ITEMs except for the first keyword:

```
VEHICLE id
      NAME Name goes here
      [DESCRIPTION textual description]
      [ATTRLIST attributes list]
      [CAPACITY int_value]
      [VOLUME int_value]
      [POSITION room_id]
      [ICON icon url]
      [PANEL panelId]
      [SHOW type]
      [IMAGE definitions here … ]
      [INNER IMAGE …]
```

For the explanation of parameters, please see *2.16- ITEM*.

Example:

```
VEHICLE car
    NAME a car
    DESCRIPTION It is a nice car
```

```
    POSITION beach
    IMAGE 240x68 auto.gif
    INNER IMAGE 350x235 console.gif
    SHOW ONSCREEN -1,24
```

Notes:
- For all VEHICLEs, the default CAPACITY is 10 and the default VOLUME is 10
- For all VEHICLES; by default the *hideable* attribute is set to 0
- The code we used for our example is provided in the **demovehicles.dxw** file, distributed with the DimensioneX kit.

2.18 SETS section

Sets section is where you can declare and initalize structured variables such as SET and ARRAY variables. INCLUDE statements could be used to include reusable libraries of SET and ARRAY definitions.

Syntax: (square brackets indicate optional parts):

```
SETS
      [SET definition]
      [ARRAY definition]
      [ … more definitions ]
END_SETS
```

SETs and ARRAYs are devices to store a number of objects in a variable by using a single identifier.

Note: They both correspond to DimensioneX's internal class named cleoni.adv.Dict, with different uses of the keys row.

2.18.1 ARRAY defined

Let's start with ARRAY, which is the simplest one. ARRAY is a container for a collection of values which you can comfortably reference via a single name. Each value has an order number in the collection with which you can reference it. This number is also referred as *key*.

This is how you define an array at design-time:

```
ARRAY id  val1[,val2 … ]
```

And this is an example in which you define a similar array at run-time:

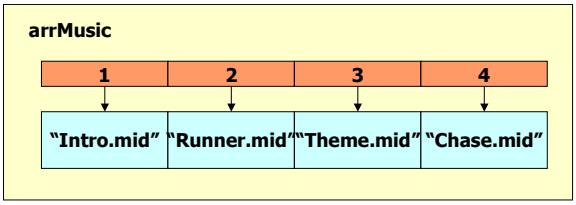
```
Function myFunction
      Dim myarr = NewArray("val1,val2,val3")
END_Function
```

Other Example

Here is a definition of an array keeping names of music clips to be played in the background:

```
ARRAY arrMusic    intro.mid,Runner.mid,Theme.mid,Chase.mid
```

Corresponds to:



In practice, the ARRAY associates **numbers** to values.
Array elements can be individually referenced via their number, example:

```
Print arrMusic(2)
```

The orange row represents the *keys* (numbers from 1 to 4) of the array, that is, information which is used to acess the content.
The cyan cells represent the *values*, that is, the content of the ARRAY.

It is important to remember the distinction between keys (=numbers) and values.

Here are some things you absolutely need to know about ARRAYs:

i) ARRAYs are indexed starting from 1

ii) ARRAYs can be extended by writing after their last element. Since the elements are numbered starting by zero, you can obtain this element number by calling the SetLen() function. This example extends the array by writing after its last element:

```
Dim possibilities = NewArray("pick,attack,go,drop,use") ' Creates the array
Print "array=" + possibilities
Possibilities(SetLen(possibilities)+1) = "bite" ' Writes after last position
Print "array now=" + possibilities
```

iii) You cannot write into an array after the last-plus-one position. If you do so, you will get an "Index out of bounds" error. Example:

```
Dim possibilities = NewArray("pick,attack,go,drop,use") ' Creates the array
Print "array=" + possibilities
Possibilities(SetLen(possibilities)+2) = "other" ' ERROR!!!
```

iv) Inside the game engine, the ARRAYs are actually implemented as SET objects in which keys are strings of the form "0001","0002",… In the present version, the maximum index for an array is 9999 (so an ARRAY can have up to about 10.000 elements).

v) The strings you specify in the ARRAY definition are always considered to be **values**. Key numbers are automatically created by the system upon seeing your ARRAY definition.

vi) By using object IDs as values (which are strings) you will also be able to create arrays referencing objects.

vii) In order to make a copy of an array, do use the **Copy( )** function. Using just the assignment operator just copies the reference to the array and/or to the elements.

2.18.2 SET defined

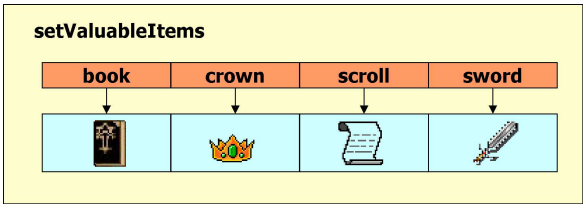
A SET is very similar to an ARRAY, but each value has a string-type *key* which is used to reference the object *value* in the collection.

Example

Here is a definition of a SET keeping ITEM objects which have a special value in the game:

```
SET setValuableItems          scroll,book,crown,sword,book
```

Corresponds to:



In practice, the SETS associates **strings** to values.

The strings you specify in the SET definition (scroll,book,crown,sword) are considered to be the **keys**. The values will be game objects referenced by the keys, which MUST have been defined.

The orange row represents the *keys* (identifiers from "book" to "sword") of the SET, that is, information which is used to acess the content.
The cyan cells represent the *values*, that is, the content.

The elements can be referenced either via their key or via their position number

```
Prints setValuableItems("crown") 'Prints the crown out
Print setValuableItems(2) ' The same, slighly faster because has direct access
```

The keys are alphabetically sorted by the keys, so keep it in mind when trying of accessing an element via its number.

Please note that in our example the keys are alphabetically sorted and do not contain duplicates of the book object.

This is how you define a SET at design-time:

```
SET id    key1[,key2 … ]
```

And this example builds a SET of strings at run-time:

```
Function myFunction
      ' Create a set that associates names to surnames
      Dim myset = NewArray("George=Clooney,Julia=Roberts,Brad=Pitt")
      Print "Here is " + myset("Julia")
END_Function
```

Storing objects with the same keys will produce overwriting.

Notes:

i) There is a way to produce unsorted sets. See the **NewSet** function.

ii) SETS can be created with the **NewSet** function, extended via a simple assignment using a new key. Elements can be removed via the **SetRemove** instruction.

iii) In order to make a copy of an array, do use the **Copy( )** function. Using just the assignment operator just copies the reference to the array and/or to the elements.

2.19 SCRIPTS section (formerly EVENTS)

Interactivity can be added by extending the game engine's default behaviour by means of custom scripts.
Scripts are the "added intelligence" to your own game.
Scripts have to be written by using DimensioneX's SmallBasic, a derivative of Microsoft VBScript language.

Your own code is called by exploiting events generated by the game engine. In fact, your code needs to be organised in EVENTs, as well as in Functions/Subroutines. All your SmallBasic code will be contained in a section named SCRIPTS (in fact it will contain also Functions and Subs).

The SCRIPTS section has the following simple structure:

```
SCRIPTS

'... one or more of the following things, in any order

[ Include "file or url" ]

[ EVENT ... ]

[ FUNCTION ... ]

[ SUB ... ]


END_SCRIPTS
```

An example follows:

```
EVENTS

      ' Events will be specified here
      ' The following one includes external SmallBasic code (optional)

      Include "mylibrary.dxl"

      ' Sample scripts follow
      ' onStart - the following one is called automatically at startup
      ' by the game engine
      EVENT onStart
            ' Things to be done at game's startup
            frontpagetext = "Welcome " + doubleNum(16)
            Call calcTimeLeft
      End_EVENT

      Sub calcTimeLeft
      ' Subroutine code
      End_Sub

      Function doubleNum(a)
      ' Compute over a
      Return a=a*2
      End_Function

END_EVENTS
```

The next chapter is a complete reference over the SmallBasic syntax to be used inside this section.

3 DimensioneX SmallBasic Reference

3.1 Introduction

SmallBasic is the language you will use to enhance the game engine's default behaviour.
Your scripts will be contained in the SCRIPTS section of your game file.
This chapter illustrates the syntax of SmallBasic, which is very similar to worldwide famous Microsoft's VBScript, but yet with several improvements.

If you are familiar with VBScript, Visual Basic or other programming languages, you will find no difficulties.
However, SmallBasic is actually even simpler than VisualBasic so it can be used also by a beginner.

3.2 Notes about SmallBasic code – very important!

- SmallBasic instructions are written one by line, and terminated by the carriage return. There's no way to put more than one instruction per line.
- The only exception of the above is for a phrase between quotes, that can contain any number of carriage returns.
- Comments can be inserted in the code by using the ' character. The rest of the line after the ' symbol will be ignored by the parser. It is recommended to place comments on a line of their own.

3.3 Include

The Include directive has the following syntax:

Include "*file*"

Use **Include** to make the game engine to load script code from an external file.
The external file, in turn, can contain other **Include**s (beware not to generate infinite include recursion, though).

File	Must be between quotes. It can be either an absolute URL (eg. https://www.myserver.com/folder/file.ext) Or a relative path. In this case, the path is intended to be relative to the folder containing the main source code. Example:

	If in C:/tomcat/webapps/dimx/WEB-INF/system/mygame.dxw I include "library.DXL" the full path of the included file is assumed to be: C:/tomcat/webapps/dimx/WEB-INF/system/library.DXL The same works for source files fetched via an URL

Note: Script Includes may not be processed at the same time as DXW World Includes (during World definition loading).

3.4 Script containers: Sub, Function, EVENT

SmallBasic scripts can be contained in blocks of one of the following types: **Sub**, **Function** or **EVENT**.

3.4.1 Sub

Use **Sub** to create a custom procedure you can call and use from anywhere. This is similar to VBScript's Sub. You can specify (optionally) a list of parameters.

Syntax:

```
Sub subroutineName[(parameter[,parameter …])]
      ' code lines here
[End_Sub]
```

The special variables $OWNER, $AGENT and $TARGET are implicitly shared with sub, and can be used there.

Example:

```
Sub mySub
Display "It works"
End_Sub
```

This sub could then be called via the statement:

```
Call mySub
```

3.4.2 Function

Use Function to create a custom procedure which returns a value. This is similar to VBScript's Function. You can specify (optionally) a list of parameters.

Syntax:

```
Function functionName[(parameter[,parameter …])]
      ' code lines here
      Return result
```

```
[End_Function]
```

The special variables $OWNER, $AGENT and $TARGET are implicitly shared with the function.

You use Functions instead of Subs when you want to calculate and return a return value. In this case, you should define a Function, and the result can be returned by using the **Return** statement. Example:

```
Function myFunction(a,b)
Display "Computing…"
Return a+b
End_Function

EVENT onNew
      Display "Calling myFunction…"
      retval = myFunction(2,3)
      Display "I'm back from myFunction - result is = " + myFunction
```

3.4.3 EVENT

You define an EVENT's code when you want your script to be **automatically run** by the DimensioneX Game Engine when something specific happens. The *eventID* you specify here needs to be exactly:

- one of the system EVENTs: read on for the full list and to understand how they work. For example, EVENT onStart is automatically called at game's startup;

one of the custom EVENTs defined in the custom PANELs defined by you (see 0 - HOOKS defines commands that the game server will accept from the outside, regardless of the state of the client (in-game player or not). Each command you specify in the HOOKS line will be associated to an event defined by you. Then, in the event code, it will be up to you what to process and what to output.

Syntax:

```
[ HOOKS hookID=eventName, … ]
```

- The function handling the event can have input parameters which will be specified by the user in the URL used for the call. The programmer may read the input parameters also via the **input()** set.
- Any output can be done via the **Print** instruction.
- The hook will be activated by opening the game's URL and using the **cmd** parameter with the specified hook ID. So you activate your event by opening an URL like this:
- http://localhost:8080/dimx/servlet/multiplayer?game=1&**cmd=hookID**&param1=foobar

3.4.4 Example

In the following example, we define a hook.

- We want to enable a way to peek into the game and quickly locate objects and characters, even when we are not connected.

- We want that, when the server receives the command: **findobj** on the game's URL, the event **doFindObj()** is triggered and, after getting the name of the object we are searching for, it outputs the results of the search in a plain text format.
- The name of the object we are searching for will be specified in the **obj** parameter.

```
GUI
      HOOKS       findobj=doFindObj  ' when you receive 'findobj' call doFindObj()
…

EVENTS
Function doFindObj(obj) 'we expect a parameter named obj
      'obj = input("obj")
      'Print "input: " + input + "<br>"
      'Print "obj: " + obj + "<br>"
      'Print "Searching: " + obj + "<br>"
      Dim c
      For Each c In getItemsIn($WORLD)
            If InStr(c.name,obj)
                  found=1
                  Print $AGENT,"<LI>" + c.name + " is in: " + c.container.name
            End_If
      Next
      For Each c In getCharactersIn($WORLD)
            If InStr(c.name,obj)
                  found=1
                  Print $AGENT,"<LI>" + c.name + " is in: " + c.container.name
            End_If
      Next
      If Not(found)
            Print "Not found: '" + obj + "'"
      End_If
End_Function
END_EVENTS
```

The command must be specified in the game's URL by using the **cmd** parameter. So you activate this hook by opening an URL like this:
http://localhost:8080/dimx/servlet/multiplayer?game=1&cmd=findobj&obj=spell

By opening the above URL you will get the result.

It should be now easy to understand that this can be used to integrate your game with any web based application.

Security notice: Be aware that when using HOOKS you open a breach into your game. While ordinary dimensioneX commands require that the user is connected or that he specifies the admin password, **any request calling your hook event will be accepted by the game server**. The task of checking who is performing the request, from what site/application, or how often is completely up to the programmer.

- PANEL definition).

otherwise the event code will never be run automatically.

Syntax:

```
EVENT eventID
      ' code lines here
[End_EVENT]
```

The special variables $OWNER, $AGENT and $TARGET are implicitly available to the EVENT code. In addition, a SET named **input( )** is automatically built by the system and includes all information collected from the user's input.

3.5 Types

As many modern scripting languages, SmallBasic does't care about types. An expression may be used as a number, a string, a boolean value depending on the context.
However, SmallBasic handles the following

3.5.1 Simple Types

numeric	integer and floating point
string	
boolean	actually implemented as numeric 1 = true, 0 = false, other non-zero value = true

3.5.2 Complex Types

image	SET object containing image objects
set	See DXW Reference->SETS section->*SET defined*
array	See DXW Reference-> SETS section->*ARRAY defined*

3.6 Operators

3.6.1 Arithmetical

Binary operators, in order of descending precedence:
^ (elevation to power eg. 2^3)
* / Mod (multiplication, division, modulus operator)
+ - (plus, minus)

Unary operator: Unary minus. Example:

a = -1

3.6.2 String

& concatenates two values as strings, regardless of their type
+ is also a binary operator and also means concatenation. Pay attention: result is a string ONLY if the first operand is a string too. Otherwise it will produce a number.

Example:

```
Print "hello" & " world!"
```

produces:
hello world!

3.6.3 Comparation

All these opartors are binary:

< less than
> greater than
= equal to
<= less or equal
>= grater or equal
<> or >< different from

3.6.4 Logical

Binary operators:

OR
AND

Unary operators:
Not( ) (is actually implemented as a function – see *3.12.39 - NOT( )*)

3.6.5 Priority

Operators have the following priority:

1 (lowest)	AND, OR
2	= < > <> <= >=
3	+ -
4	* / ^
5	. (indirection)
(top)	- (unary)

To change priority, use round brackets. Example:
a = 3 + 2 * 5
Above, 2*5 is evaluated first because * has the highest priority with respect to +. To change:
a = (3 + 2) * 5
In this case 3+2 is evaluated first because of the brackets.

3.7 Variables and properties

3.7.1 Defined

- The game engine can "remember" values you stored into *variables*. Attributes of existing objects are said *properties*, but they are in practice the same thing as variables. We could even say that variables actually *are* properties of the WORLD object.
- Variables are immediately accessible to all objects by specifying their name. Example:

```
Print mode
```

  While for object's properties you need a reference to the object first:

```
Print obj.name
```

- Properties/variables need not to be declared, and are automatically created the first time they are assigned a value. It is however recommended to declare and use local variables in scripts.
- See also *3.9.1 - assignment: [object.]property = expression*

3.7.2 Declaring Variables

- It is a good habit to declare variables you use temporarily in custom functions, events and subroutines.
- You can declare a local varibale and even assign a value to it by using the **Dim** instruction.
- The visibility of the local variable is limited to the **Function**/**Sub**/**EVENT** in which it is declared.
- Local variables are protected and allow the functions to call themselves recursively without any harm to local values.
- Some properties are already defined and you can use them for your own purposes, they are documented below in the *Object models*.

3.7.2.1 Examples

The following assignment:

```
temp = 1
```

Assigns the value 1 to the variable named *temp*. The scope of the variable (local, global) depends on either or not the variable was declared using the **Dim** instruction.

The instruction:

```
Display temp
```

Displays on the user's console the value you previously stored, that's the number 1.

Consider the following:

```
surfer.health = surfer.health - 1
```

The above line of code reads the *health* property of the *surfer* character, decrements it, and then stores the result in *surfer.health*, again.

3.7.3 Special variables

Some variables cannot be assigned and are automatically set by the system. You should never attempt to assign them any value, but you can read their value in your scripts.

True	always equal to 1
False	always aqual to 0
Null	Always equal to ""
SYS	Refers to the game's default SYSTEM voice. It can be used as a default neutral speaker.
$WORLD	Represents current world.
input( )	This is a SET containing key/value pairs from the user's command PANEL. This is actually the input of the user. **Note:** This set has got values only when read in custom events and for the **onLook** event.

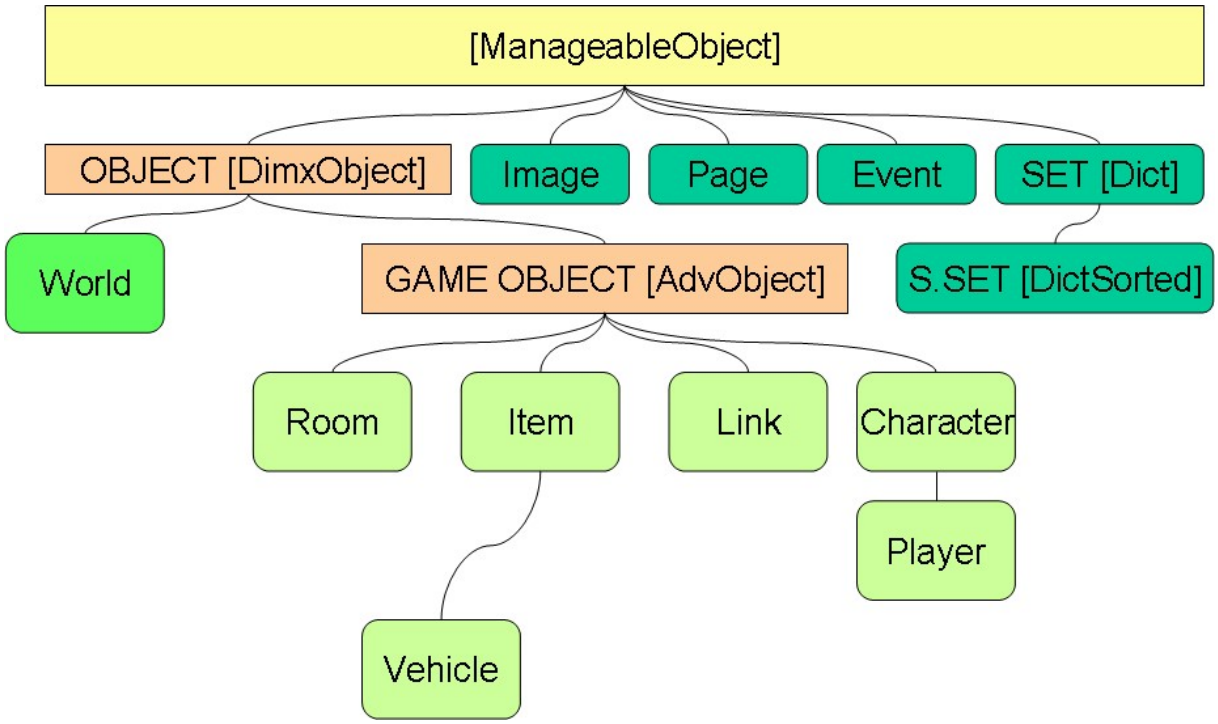
In addition, all the following identifiers can be used in EVENT scripts: $AGENT, $OWNER, $TARGET. For their meaning, read *3.10.1.3 - EVENTs parameters*.

3.8 Object models

This section tells you about important properties of the several game objects. Please note that most of them (CHARACTER,ITEM,ROOM,LINK) share common features so there is a chapter that contains information that is pertinent to all, then you have to look at the type-specific section.

3.8.1 Class Hierarchy

The following diagram shows the class hierarchy of the scripting engine, which clearly maps over the underlying class hierarchy of the Java engine. The names of underlying Java classes are written in square brackets.



3.8.2 Object model for: WORLD

property/variable name	special limitations	description
bgcolor		Background color for the sky in transparent scene images (any valid html value such as "black" or "#FFGGEE"). If null the default color for the sky will be: #9DB9E9
debugmode	Read only	1 or 0, according to the debugMode value set in the worldnav.properties file
interphone		Interphone level. See section about DXW Syntax - WORLD for details
instanceid	Read only	Unique string representing the world instance
frontpagetext		String to be displayed at the game's logon screen. May include HTML code.
mode		Initially equals to **null**, is set to zero (0) with the **Goal** instruction. It is suggested to set this variable to 1 at game's startup to properly detect when the game was won.
name	Read only	Title of the game

_optiRuntimeErrors		Records the number of runtime errors occurred so far.
_optiFrequency		SET object containing the frequency of calls for each Sub/Function/EVENT
_bannedClients		Pipe-separated list of IP numbers to be banned.

3.8.3 Object model for: any GAME OBJECT

3.8.3.1 Properties

These properties are common to ROOM/ITEM/CHARACTER/LINK objects. See also **Object-specific Object models** below for additional properties!

property/variable name	**special limitations**	**description**
id	read only	Identifier of the object.
name		Name of the object.
description		Is displayed when the player looks at it. A description starting with an ampersand sign '@' means "dynamic description": the part following the ampersand has to be considered as a function: it will be executed and its result will be used as the object's description.
type	only for ITEMS, CHARACTERS	Type identifier of the object
capacity	only for ITEMS, CHARACTERS	Capacity, in units
volume	only for ITEMS, CHARACTERS	Volume, in units
volume	only for ITEMS, CHARACTERS	Volume, in units
container	-read only **-to change**: use the **Move** instruction -Cannot be used on LINK objects and on ROOM objects	Reference to the container object.
facing	only for ITEMS, CHARACTERS	facing direction: letter: N/S/E/W
hidden	only for ITEMS, CHARACTERS	Is the object hidden? Boolean value.
invisibile	Not available for	Is the object invisible? Boolean value. Use it only for

	ROOM	"onscreen" objects
icon	Not available for ROOM	Is the URL to the object's icon. URL can be absolute, or relative to the WORLD's IMAGEDIR.
image		Is a SET containing IMAGE objects, that are all the images for the object. The set keys are the directions "N","S","E","W". The perfect and safest way to assign values would be: object.**image(***face***)** = NewImage("*url*",*width*,*height*) *face* can be any of the following: **"**W" |" E" |" S" |"N" But the game engine also accepts: object.image(*face*) = "url" in which case the url of the specified image gets simply replaced, the dimensons remain unchanged. The game engine also accepts: object.image = "*url*" or object.image = NewImage(…) which is not correct, but works **if and only if** the object has got just one image already, which will be updated.
showmode	For ITEM, CHARACTER, LINK only. Write only.	Defines how the object will show on the screen. See also: *2.9.1 - showMode*. 0 (means OFFSCREEN) 1 (means ONSCREEN) 2 (means ICON)
showX, showY	For ITEM, CHARACTER, LINK only. Write only.	Defines where the object will be positioned on the scene. See also: *2.9.2 - showX, showY*.
showFor	For ITEM, CHARACTER, LINK only. Write only.	Defines the scene image to which the object is linked. See also: *2.9.3 - showFor*.
any other identifier		Generic, user-defined, object property

3.8.3.2 Methods

These methods can invoked any game objects objects which correspond to human players.

3.8.3.3 OBJECT.getProperty(property)

Returns the object's specified *property*. It provides a convenient way to access arbitrary properties to be defined at run time.

```
x = item.getProperty("name")
```

is equivalent to:

```
x = item.name
```

See also: *setProperty* method.

Parameters

property String representing the property to be inspected

3.8.3.4 OBJECT.setProperty(property,value)

Sets a new value to an object's specified *property*, then returns it. It provides a convenient way to access arbitrary properties to be defined at run time.

```
Call item.setProperty("Value",100)
```

is equivalent to:

```
item.Value = 100
```

See also: *getProperty* method.

Parameters

property String representing the property to be assigned a value
value Value to be assigned

3.8.4 Object model for: ROOM

3.8.4.1 Properties

These properties are specific to ROOM objects. See also **Object model for any GAME OBJECT** above.

property/variable name	special limitations	Description
mapx, mapy		Specifies the co-ordinates of the ROOM on the game's map. This is used to track the player's position automatically. If you don't specify it, the position will

		simply not be updated when the player walks in the room.
map		Alternate map to be used for this room, instead of the WORLD's default. Specify either an url or an image name, referring to the default image folder.
effect		If contains a valid IMAGE object, this one is displayed on top of ROOM picture and below all other objects. Ideal for animated effects (moving rain, mist, wind). Use it in conjunction with transparent images (GIF or PNG formats). **Positioning:** Effect image is placed starting from the highest point in the scene picture.

3.8.5 Object model for: CHARACTER

3.8.5.1 Properties

These properties are specific to CHARACTER objects. See also **Object model for any GAME OBJECT** above.

property/variable name	**special limitations**	**description**
accepts	-write only	equivalent to the CHARACTER.ACCEPTS tag. Can be any of the following values: - "*" (all) - "" (nothing) - a comma-separated list of item IDs and/or TYPE identifiers
__clearfocus		If set to TRUE, cancels the focusing on the currently selected object. Please note this property begins with two underscores ("_")
__clearinfo		If set to TRUE, cancels the display of the information box about object's properties. Please note this property begins with two underscores ("_")

3.8.5.2 Methods

3.8.5.3 CHARACTER.go(*direction*)

Moves the character in the specified *direction*.

3.8.6 Object model for: PLAYER

3.8.6.1 Properties

These properties are specific to PLAYER objects.

Note that a PLAYER is also a CHARACTER and a GAME OBJECT so see also **Object model for any GAME OBJECT** and **Object model for CHARACTER** above.

property/variable name	Notes	description
remoteAddr		IP number of the player's client
mute		When WORLD.MUTING is on, indicates the number of time the player has used a forbidden word

3.8.6.2 Methods

These methods can only invoked on CHARACTER objects which correspond to human players. If in doubt, use the **IsPlayer**() function to make sure that the object is a player.

3.8.6.3 PLAYER.getCookie(key)

Retrieves a setting from the player's browser cookies. Returns the value associated to the specified *key*. See also: *saveCookie* method.

Parameters

key Key to be seached in the database

3.8.6.4 PLAYER.getPanel()

Returns the ID of the player's current PANEL.

3.8.6.5 PLAYER.printXY(stuff,x,y)

Prints stuff at x,y on the player's scene image. If the current ROOM has no image, no printing is d.

Parameters

stuff Stuff to be printed. An image, or a String (HTML is OK)

x,y Co-ordinates to be used. Note: using negative values for x,y or values that fall outside the scene image's boundaries may lead to unpredictable results.

3.8.6.6 PLAYER.saveCookie(*key,value*)

Saves a setting into the player's browser cookies. The specified *value* will be associated to the specified *key*. See also: *getCookie* method.

Parameters

key Key to be associated to the value

value Value to be stored (string)

Warning!: The **saveCookie** method steadily increases the amount of information that is transmitted by the client to the server. For this reason, it is recommended to use it only for really relevant information

3.8.7 Object model for: ITEM and VEHICLE

3.8.7.1 Properties

These properties are specific to ITEM objects. See also **Object model common to ROOM/ITEM/CHARACTER…** above.

property/variable name	special limitations	description
innerImage		IMAGE corresponding to the object's inner image.
zoomImage		IMAGE corresponding to the object's zoom image.
showPolicyPeople		Possible values are (SHOWMODE codes): 0 (default) means OFFSCREEN – all contained characters will be shown off screen as icons 1 means ONSCREEN – all contained characters will be shown on screen

3.8.8 Object model for: IMAGE

property/variable name	special limitations	description
url		URL of the image
width		Width of the image
height		Height of the image

See also notes on the **image** attribute in section 3.8.3 for important, additional information.

3.8.8.1 Methods

3.8.8.2 IMAGE.html(*title[,player]*)

Outputs the image in HTML.

title Optional. ALTernate text for the image (tooltip)
player Optional. If specified and if it references a valid player, the image is automatically resized according to his/her browser's resolution, as it it done for the scene image.

3.9 Constructs, Flow Control

These are classic instructions constructs which can be found in other programming languages. They also can be used in SmallBasic.

3.9.1 *assignment: [object.]property = expression*

Assigns a value to a variable (syntax 1 below) or to an object's property (syntax 2 below).
Syntax:

variable = ***expression*** (syntax 1)

object.property = ***expression*** (syntax 2)

Parameters:
object Any object identifier.
variable, property Any variable/property identifier.
expression Any valid expression.

You can use multiple indirection operators (“.”). Example:

```
$AGENT.container.name = "Mystery object"
Display $AGENT.container.name
```

is valid and displays the name of the container of the object having caused the latest event.

Important!
Rule#1: Reading a property from a NULL object always yelds a NULL result and produces a WARNING on the log.
Rule#2: Attempt to assign to any property of a NULL object produces an ERROR.

3.9.2 If *condition* … (Else …) End_If

Represents the classic IF or IF … ELSE construct

The *condition* is evaluated. If evaluates to TRUE the instructions block following the IF and preceeding the Else or the End_If is executed.
If the condition evaluates to False and an Else block is present, it gets executed.

example

```
If $AGENT.Health > 0
      Speak "You look healthy!"
Else
      Speak "You look awful… and now you die!"
      Kill
End If
```

Note: If .. End_If constructs can be nested one into another

3.9.3 For *idx = startVal* To endVal … Next

Represents a classic loop construct based on iteration. An index variable *idx* takes all values between startVal and endVal, and the block of code before the *Next* keyword is executed each time.

In the following example, a **For .. Next loop** is associated to the onTick event so that 10 numbered messages are displayed (onTick triggers every 30 seconds). This loop can be used to dynamically create a number of objects.

```
EVENT onTick
      For i=1 To 10
            Display "Looping " + i
      Next
```

Note: startVal should normally be less or equal to endVal. If endVal < startVal then a waring is produced on the log.
Important! For … Next constructs cannot be nested one into another in this release. If you need to do this, define the inner For cycle in a seperate Sub and call the Sub from inside the outer For cycle.

3.9.4 For Each *key* In *set* … Next

Represents a loop construct with iteration over a set of values. An index variable *key* takes all values of the keys in the specified *set*, and the block of code before the *Next* keyword is executed each time.

In the following example, a **For Each** loop is used to browse the set of object contained in a room (resulting from the function **getItemsIn**) and to decide which one is to be picked up.

```
      ' Monsters pick up coins
      Debug $OWNER.name + " is in " + $OWNER.container.name
      tobepicked = ""
      For Each item In getItemsIn($OWNER.container)
            If Not(item.hidden) And item.Cash > 0
                  tobepicked = item
                  Debug $WORLD,$OWNER.name + " sees " + item.name
            End_If
      Next
      If tobepicked <> ""
            Move tobepicked,$OWNER
            Debug $WORLD,$OWNER.name + " has picked up: " + tobepicked.name
      End_If
```

Important! For … Next constructs cannot be nested one into another in this release. If you need to do this, define the inner For cycle in a seperate Sub and call the Sub from inside the outer For cycle.

3.10 System EVENTs

To allow true interactivity, DimensioneX uses system events like programming languages do. For example, when you give an object to an actor, the event **onReceiveItem** is triggered for that actor.

Then, use these events as explained here.

A typical EVENT specification is as follows:

```
SCRIPTS

EVENT event_qualifier
[ACTIONS]
[     SmallBasic instructions follow here
      ... ]
[END_EVENT]

END_SCRIPTS
```

ACTIONS
This keyword is optional and indicates the beginning of the EVENT's code. Place SmallBasic instructions one after another, one line for each one.

3.10.1 The DimensioneX event model

Prior to looking at the events syntax, you have to understand the **DimensioneX event model**, which is somehow similar to that used by MS Visual Basic.

The *DimensioneX event model* is a method that is used to implicitely define the input parameters for each system EVENT.
By knowing each system EVENT, you know which special variables to examine ($AGENT,$OWNER and $TARGET) to get the input parameters.

Please note that user-defined, custom events also receive the user's input in a SET named **input**, which makes this EVENT model rather useless in this special case.

3.10.1.1 Basic principle

- When something happens (i.e. a player picks up an item), the game engine triggers an EVENT with a specific name (say, *item.whenPicked*), depending on what happened.
- Then the game engine checks if in the SCRIPTS section an associated EVENT script exists (thus declared as "EVENT item.whenPicked"), then the associated code is executed.

So:
- the association (thing happening) → (event fires) is implicit and managed automatically by the system.
- The association (event fired) → (script execution) is also implicit but can be controlled by the programmer, by using the correct EVENT definition for each event he/she wants to associate scripts with.

This technique is commonly used also in Visual Basic so programmers should have not problems in understanding it.

3.10.1.2 Figuring out EVENTs

To understand what events are fired in consequence to what action is easy.

Suppose we want to know what EVENTs are triggered when I pick up an item. Then, just do the following:

1. Start your game, log in
2. Pick up the item, as supposed
3. Check the **debug.log** file. You will see the events noted there. You will find something like:

```
USER (_p0) STATUS: CONNECTED COMMAND: pick(i!cross,) VIEW: scene COUNT: 16
world.fireEvent: event: cross.onExit(owner=cross (a cross),agent=_p0,target=_p0)
world.fireEvent: event: onExit(owner=cross (a cross),agent=_p0,target=_p0)
world.fireEvent: event: chapel.onLooseItem(owner=chapel (the chapel),agent=_p0,target=cross)
world.fireEvent: event: onLooseItem(owner=chapel (the chapel),agent=_p0,target=cross)
Sending: refresh!scene in chapel
world.fireEvent: event: cross.whenPicked(owner=cross (a cross),agent=_p0,target=_p0)
world.fireEvent: event: whenPicked(owner=cross (a cross),agent=_p0,target=_p0)
world.fireEvent: event: _p0.onReceiveItem(owner=_p0 (Cris),agent=_p0,target=cross)
world.fireEvent: event: onReceiveItem(owner=_p0 (Cris),agent=_p0,target=cross)
EXIT SERVICE
```

In the above example, **onExit**, **onLooseItem**, **whenPicked**, **onReceiveItem** are the EVENT names. Note that any event is related automatically to 3 parameters: owner, agent and target (also defined as *Roles*). See the section below for an explanation on these.

4. Now you can read the EVENT specifications below and try to understand what EVENT better suits your needs. Then you simply write an EVENT script in the SCRIPTS section of your game to exploit the event you chose.

3.10.1.3 EVENTs parameters

For each event, event-related objects will acquire a role, and you can reference them according to their role, as if they were pre-defined parameters.

Here you have a generic table explaining these roles:

$OWNER Is the object which ***owns*** the event. That's to say, the object to which the event is directly associated.

Example:
consider the event: **surfer.onLook**
surfer is the OWNER of the onLook event.

$AGENT Is the object that caused the triggering of the event.

Example:
consider the event: **surfer.onLook**
if a player looks at the surfer, he will become the event's AGENT.

$TARGET Is another object involved in the event (example: an ACTOR uses X on Y: well ACTOR is the *agent*, X is the *owner* of the onUseWith event, and Y is the *target*).

Example:
consider the event: **key.onUseWith**
if a player uses a key on a door, that door will become TARGET.

In other words, each time an event occurs, three parameters will be instantiated so that your scripts can reference them (see also 3.7.3 - Special variables – page 71) in order to produce a meaningful reaction.

See the next sections explanation of all the supported events and their roles/parameters.

If in doubt about what parameters will be used, please adopt the technique explained above in *3.10.1.2 - Figuring out EVENT*.

3.10.1.4 EVENTs Return value

Please note that all event scripts, by default, will be considered as having returned a success result when their code execution completes.
In some specific EVENTs (where indicated) a zero value can be returned via the **Return** statement. In that case, the associated action will be automatically cancelled. The game engine may prints out messages accordingly.

3.10.2 beforeOpen

Triggered before an item is opened.

Applies to: ITEM
OWNER: Item being opened
AGENT: Character opening the item
TARGET: unused
Ideas for use: Unlocking of items based on keys.

3.10.3 Living

Triggered each 30 seconds, for each CHARACTER in the world, individually. The event's purpose is to decide whether the character should continue to exist.
If false (0) is returned, $OWNER dies. if true (non-zero) is returned, $OWNER lives for 30 seconds more.
Living triggers after the **onTick** event.

$OWNER: The CHARACTER whose life condition is being evaluated.
$AGENT: The CHARACTER whose life condition is being evaluated.
$TARGET: The CHARACTER whose life condition is being evaluated.
Ideas for use: Survival condition, robots' movements

Example:

```
EVENT Living
ACTIONS
    ' Restores health a little bit
```

```
    $OWNER.Health = $OWNER.Health + 0.2
    If $OWNER.Health > 10
        ' Limits health to 10
        $OWNER.Health = 10
    End_If
    'Returns result - if positive, character will live
    Return $OWNER.Health > 0
```

3.10.4 onClose

Triggered when an item is closed

Applies to: ITEM
OWNER: Item being closed
AGENT: Character closing the item
TARGET: unused

3.10.5 onDbDown

Notifies the game engine that the database connection cannot be established. This is used when a mySQL database has been configured and it can be useful to avoid that players go on thinking they are using the most updated data. In this case, in fact, the game will be using the data stored on the text .SAV file, so data will not be up-to-date.

When the connection with the database will be restored, an onDbUp event will be triggered automatically.

Applies to: No object.
OWNER: Unused
AGENT: Unused
TARGET: unused

3.10.6 onDbUp

Notifies the game engine that the database connection has been successfully established. This is used when a mySQL database was in use, the connection was lost (onDbDown event) and it can be useful to inform players that the system has returned to work nomally. After this event, in fact, the game will be using the data stored in the database, so data will be up-to-date.

Applies to: No object.
OWNER: Unused
AGENT: Unused
TARGET: unused

3.10.7 onDie

Triggered when a player dies

Applies to: No object.
OWNER: Player dieing
AGENT: Unused
TARGET: unused

3.10.8 onEnter

Applies to any object entering another object not being a CHARACTER.

Applies to: moveable objects (ITEM, CHARACTER, VEHICLE)
OWNER: object entering
AGENT: Object causing the action.
TARGET: Where/what OWNER is entering

Notes:
- Triggers right before **onReceive**. Triggers after the movement has actually happened.
- If the entered object is a CHARACTER, then **whenPicked** EVENTs wil trigger instead

3.10.9 onExit

Symmetrical to onEnter, applies to any object exiting from any object not being a CHARACTER.
OWNER: object exiting
AGENT: Object causing the action.
TARGET: Where/what OWNER is exiting.

Return value: is 1 by default. If 0 is returned, the action gets cancelled.
Notes:
- Triggers right before **onLoose**.
- If the exited object is a CHARACTER, then **whenDropped** EVENTs wil trigger instead

3.10.10 onHear

Triggered when a character receives a message.

Applies to: any object except for $WORLD
Returns: True by default. If the result is set to False, then the action is cancelled.
OWNER: Object hearing the message
AGENT: Character having spoken the message
TARGET: Text of the message
Ideas for use: Computer-controlled actors which act in response to your questions, text-driven user input

Note: If the named version of the event has been defined (ID.onHear) then it is executed, while the generic version (onHear) is not fired so in case you need it, call it explicitly. If the named version (ID.onHear) does not exist, the generic version of the event is fired (onHear).

3.10.11 onLoose

Applies to any object, acting as a continer, is about to lose any CHARACTER object.

$OWNER: Container object.
$AGENT: Object causing the action. Typically who is moving the object away from OWNER. If it is a character exiting the game, this will be equal to null.
$TARGET: ID of the destination link OR ID of object being moved.

- If the movement is caused by the system (i.e. because a character dies and it is removed by the system) then AGENT = NULL and TARGET = object being moved.
- If a CHARACTER is about to traverse a LINK, AGENT = character moving, TARGET = LINK being used.

Return value: True by default. If False is returned, then the action gets cancelled.

Note: Triggers after onExit

3.10.12 onLooseItem

Same as for onLoose, except that triggers when OWNER is losing an ITEM or VEHICLE object.

Return value: is 1 by default. If 0 is returned, the action gets cancelled.
Note: Triggers after **whenDropped** and before **whenPicked**

3.10.13 onLook

Triggered when an object is looked at

Applies to: ITEM, CHARACTER, ROOM
OWNER: Object being looked at
AGENT: Character looking
TARGET: unused
Return value: True by default. If the result is set to False, then the action is cancelled.
Note 1: If you just wish to skip the focusing effect, just set the player's **.__clearfocus** property to True.
Note 2: The individual OBJECT.onLook event triggers first than the generic onLook EVENT. If the individual OBJECT.onLook exists and if its return value is not explicitelty zero, then the event is considered as handled and the generic onLook event is not triggered.
Note 3: With difference regards all other system events, this event receives user input in the **input( )** set.
Ideas for use: Characters speaking or talking to players when they look at them, giving hints on the game.

3.10.14 onNew

Is triggered when a new player enters the world.

Applies to: No object. It refers to the whole game
AGENT: The Character entering the world.
TARGET: null
OWNER: null

Ideas for use: Voices speaking or sounds playing when a player joins.

3.10.15 onOpen

Triggered when (after) an item is opened.

Applies to: ITEM
OWNER: Item being open
AGENT: Character opening the item
TARGET: unused
Ideas for use: Traps inside boxes, explosive bags.

3.10.16 onReceive

Applies to any object, acting as a continer, is about to receive any CHARACTER object.

OWNER: Container object.
AGENT: Object causing the action.
TARGET: What is being received or placed in OWNER. The LINK won't appear here as it happens for onLoose.

Note: Triggers right after **onEnter**.

3.10.17 onReceiveItem

Same as for onReceive, except that triggers when OWNER is receiving an ITEM or VEHICLE object.

Return value: if TRUE (default) the action is considered as successful. If FALSE, the action is considered failed, but it is programmer's responsibility to actually MOVE the object away if desired.

Note: Triggers right after **whenPicked**.

3.10.18 onSave

Triggered when a player is saving his game, right before the save operation.

Applies to: Player saving the game

OWNER: Player
AGENT: Player
TARGET: If "Save" was chosen: Player, if "Save and exit" was chosen: null string ("")

Return value: is 1 by default. If 0 is returned, the user command (save or save+exit) gets cancelled.

Ideas for use: Check and avoid player to save too often

3.10.19 onSearch

Triggered when an item is hidden, and a **search** command is performed. This event is deisgned to implement selective search. If **false** or 0 is returned, the item will not be found. Otherwise the item will be found.

Applies to: ITEM

OWNER: The item about to be found
AGENT: The character searching
TARGET: not used
Return value: TRUE by default, which means "yes, the item can be found". If FALSE, then the item whose id is *ID* will not be found.

3.10.20 onSpeechFinish

Triggered when a computer-controlled actor has finished its speech done via the SPEAK "*" action (See SPEAK for details).

Applies to: CHARACTER
OWNER: Computer-controlled CHARACTER speaking
TARGET: CHARACTER receiving the message
AGENT: same as above
Ideas for use: Computer-controlled actors which speak to players and at the end of the speech give an item

3.10.21 onStart

Triggered whenever the world starts. It is triggered also upon reset.

Applies to: No object. It refers to the whole game
OWNER: not available
AGENT: not available
TARGET: not available

Ideas for use: Put any initialisation code here (setting of global variables, etc.)

3.10.22 onTick

Triggered whenever the internal clock generates a "tick". This happens **every 30 seconds**.
onTick triggers right before the **Living** event.

Applies to: No object. It refers to the whole game
OWNER: not available
AGENT: not available
TARGET: not available
Ideas for use: Any changes proportional to elapsed time, brief pauses

3.10.23 onUse

Triggered when an item is used (without secondary target – example "USE switch").

Applies to: ITEM
OWNER: The item being used
AGENT: The character having used the item
TARGET: not used
Ideas for use: Switches, controls in general

Note: If the named version of the event has been defined (ID.onUse) then it is executed, while the unnamed version (onUse) is not fired. If the named version (ID.onUse) does not exist, the unnamed version of the event is fired (onUse).

3.10.24 onUseWith

Triggered when an item is used over another item (example "USE + key ON door")

Applies to: ITEM
OWNER: Item being used (key in the above example)
AGENT: Character using the item
TARGET: Item on which we use the OWNER object (door in the above example)
Ideas for use: Unlocking doors or other objects, attacks to players and monsters

Note: If the named version of the event has been defined (ID.onUse) then it is executed, while the unnamed version (onUse) is not fired. If the named version (ID.onUse) does not exist, the unnamed version of the event is fired (onUse)

3.10.25 restore(type,restoreinfo,player) : boolean

This event is triggered by the system each time an object is being restored during a player logon.

Parameters:

type: TYPE of the object being restored
restoreinfo: Additional restore information as calculated by the **saveInfo** event (see)
player: Player who is logging back on
OWNER: null (unused)
AGENT: null (unused)
TARGET: null (unused)
Return value: Result of the operation. **true** means 'object restored'; **false** means 'object NOT restored'. In the latter case, the system will proceed to try an auto-restore.

Note: this event is generic and cannot be object-specific.

Example: See saveInfo( )

3.10.26 saveInfo( ) : string

This event should be used to produce the information which is relevant for a given object to be restored later. This information will be saved together with object ID and TYPE, and can be used to reconstruct the object when restoring the game profile (this should be done via the restore( ) EVENT).

Applies to: any object being in the player's inventory when player issues SAVE AND EXIT. If the player has just cliked SAVE, then this event does not fire.
OWNER: object in inventory
AGENT: null (unused)
TARGET: null (unused)
Return value: Any string value representing relevant information about the object. Default return value is **null**.
Notes: this event is object-specific and cannot be used as generic.

Example:

The first event simply contains the code to extract the relevant object's information, which is the item's Cash property. The restore function simply reconstructs the money .

```
Event saveInfo()
      type = $OWNER.type
      Dim txt
      If type = "money"
            txt = ""+$AGENT.name+" saves money"
            'Print $WORLD,txt
            'Debug txt
            Return $OWNER.Cash
      End_If
End_Event

Event restore(type,restoreinfo,player)
      If type = "money"
          Print player,"restore() restoring: type:" + type + " info: " +
restoreinfo + " player: " + player.name
      Dim value = restoreinfo
      Dim newobj = NewItem(target,"" + value + " coins","Gold coins are
money!",NewImage("money.gif",31,31),"volume=0,type=money,pickable,showmode=2,icon=m
oney.gif,Cash="+value)
      AttachEvent newobj,"saveInfo","money_saveInfo"
      Return true
    End_If
    return false
End_Event
```

3.10.27 whenPicked

(formerly onPick)
Applies to any object entering a CHARACTER. Similar to onEnter but in this case the container is a CHARACTER.

Applies to: moveable objects (ITEM, CHARACTER, VEHICLE)
OWNER: object entering (being picked)
AGENT: Object causing the action.
TARGET: Who is receiving OWNER

Notes:
- Triggers right before **onReceiveItem**.
- Triggers **after** the item **has been actually picked up**. If the action needs to be cancelled, before returning False be sure to **Move** the object somewhere else.
- If the entered object is not a CHARACTER, then **onEnter** EVENTs wil trigger instead

Ideas for use: Magic objects, Armories, Traps

3.10.28 whenDropped

(formerly onItemDrop)
Symmetrical to whenPicked, applies to any object exiting from any CHARACTER object.

OWNER: object exiting (being dropped)
AGENT: Object causing the action. Attention – might be equal to NULL when using Move instruction
TARGET: New destination for OWNER.

Return value: is 1 by default. If 0 is returned, the action gets cancelled.
Notes:
- Triggers right before **onLoose**.
- If the exited object is not a CHARACTER, then **onExit** EVENTs wil trigger instead

Ideas for use: Magic objects, Armories, Traps

3.11 Instructions (also defined Statements or Actions)

When an event triggers, you can tell DimensioneX to perform several actions.
Each action usually acts on the objects which the event was originally related to.
Actions are specified in a language which is called SmallBasic. Programmers familiar to Visual Basic or other programming languages should have no difficulties in using it.

Parameters specify how action is to be performed. Parameters' meaning and availability depend closely on the action you want to perform, so check action documentation carefully.

3.11.1 AttachEvent *dest, "eventId" , "attachedEventID"*

Attaches an event-management script to an object.

Parameters:

dest ID of the game object to which the EVENT script is being attached. It must be a valid object. Cannot be equal to $WORLD

eventID Event ID which will be enabled after the execution. This must a plain ID, i.e. it must not not contain any qualifier dot (in other words: *"object.idstring"* is not allowed but *"idstring"* is).

attachedEventID Event ID from which the code is being taken. When attaching code to players, it is not allowed to attach another object's event (a WORLD's event must be used instead). A null string will cancel any event corresponding to eventID.

Example: We have en event named **onEnter_Vampire** which should be applied to a character and check whether he/she is walkig in sunshine. In the case, the character dies. The following line attaches the code of the event **onEnter_Vampire** to the character referenced to by **person**, so that it will be executed upon triggering the system event **infected.onEnter**. This will avoid using a generic event **onEnter** or all characters, and make execution more efficient.

```
AttachEvent person,"onEnter","onEnter_Vampire"
' Now, each time person enters a room, the code taken from onEnter_vampire will be
' executed
AttachEvent person,"onEnter",""
' Cancels the previous one
```

3.11.2 Ban *player*

Bans the specified *player*.

Parameters:

player (Required) ID of the player to be banned.

3.11.3 Call *subroutine*

Calls the specified *subroutine* or event, implicitely passing owner, agent and target.

```
Call subroutine([parameter list])
```

Parameters:

subroutine (Required) Subroutine or event identifier

parameter list (Optional) parameters of the subroutine

3.11.4 Dim

Declares and creates a local variable. The variable is linked to the variable space of the function, event or Sub in which the declaration is made. **Dim** can be used with any of the following syntax forms:

```
Dim variable
```

Creates the *variable* which is specified after the **Dim** keyword. *variable* is filled with a *null* value.

```
Dim variable=value
```

Creates the *variable* which is specified after the **Dim** keyword. *variable* is filled with the specified *value*.

```
Dim var1,var2=value,var3
```

The same for more than one variable at the same time.

Note 1: Any local variable is protected and safe from use across recursive calls.
Note 2: Local variables are not visibile from outside, nor from called functions/subs
Note 3: Global variables cannot be declared by using Dim, which always creates a local variable. For creating global variables, simpy assign to them a value (they are created dynamically).

3.11.5 Debug *message*

Sends the specified message to the debug device (normally the **debug.log** file in the dimx system folder).

Parameters:
message (Required) Message to be displayed.

3.11.6 Display *[dest,] message [, message …]*

Synonym for Print. See *3.11.18 - Print [dest,] message [, message ...]*.

3.11.7 DropItems *victim*

Make the specified *victim* object to drop all the carried items. If the target is a player, a message is also displayed on the console.

Parameters:
victim (Optional) ID of the container object. If omitted, the default is $AGENT.

3.11.8 For *idx* = *startVal* To endVal … Next

Represents a classic loop construct based on iteration. An index variable *idx* takes all values between startVal and endVal, and the block of code before the *Next* keyword is executed each time.
See section *3.9.3 - For idx = startVal To endVal ... Next*.

3.11.9 For Each *key* In *set* … Next

Represents a loop construct with iteration over a set of values. An index variable *key* takes all values of the keys in the specified *set*, and the block of code before the *Next* keyword is executed each time.
See section *3.9.4 - For Each key In set ... Next*.

3.11.10 Goal *[message]*

No parameters. Has the following effects:

- Sends out a broadcast *message* (if no message is specified, the default one is used) usually telling that the game has been finished, when, and by whom.
- A global variable named "**mode**" (the programmer is supposed to have set its value to 1 at the game start) is set to zero.
- The event is recorder in the game's Hall of Fame (gamename.hof in the **system** folder).

Tip: Checking the content of the global variable named **mode** can be a good means to detect when the game has been finished at run-time.

3.11.11 Journal title,link,text,categories

Creates a new entry on top of the game/cluster's journal file, if configured.

A journal file is a sort of embryonal form of RSS (e.g. it can be easily converted to a standard RSS format) and will contain the 20 most recent entries.

If no journal file has been configured for the current game or cluster, no result is produced.

See section ***1.2.14 - Output to RSS feed – integrating with Facebook, Twitter and other social networks*** for more information about Journal files and producing RSS feeds.

Syntax:

```
Journal title,link,text,categories
```

All parameters are mandatory and must be strings.

- **title** is the entry's title, it will become the RSS feed item's clickable title
- **link** is meant to become the RSS feed item's URL associated to the title. It can be either your game's official web page, the player's profile page, whatever you think it's appropriate
- **text** is the text of the entry, can contain images and line breaks coded as \n (backslash, n). If you insert carriage returns as Chr(13) they will be automatically converted by the system.
- **categories** is a string made of a single category OR however-you-want separated category slugs of your choice, for your RSS feed converter's sake. This info has been designed so that advanced users using the RSS feed will be able to filter out the info they want by using their own feed reader's controls (eg. one may want to get notified about just events regarding HIS profile)

Note: No parameter, except for **text**, can include carriage returns, otherwise your converter to RSS will have troubles in parsing the journal file. This is, however, out of the scope of dimx.

Example:

```
Journal "Restart",gameInfo("site"),"Game has been restarted.\nYou can connect and
play now.","news,system events"
```

3.11.12 Kill *[victim]*

Removes an object from the current world.

Parameters:

victim (Optional) ID of the object to be removed. If omitted, the default is $AGENT.

3.11.13 Move *[what], where*

Moves an object.

Parameters:
what (Optional) What is going to be moved. Defaults to $AGENT
where Object (ROOM or ITEM) where it is going to be moved.

3.11.14 MoveOutside *what, areaid*

Moves an object to another WORLD.

Parameters:
what What is going to be moved.
areaid Id of the area (WORLD) where the object is being moved to. This one must belong to the same CLUSTER as the current one.

3.11.15 NewRoom *id*

Creates a new ROOM object with the specified id. Other object features such as name, description and image must be set later by using assignments (see also *3.8.3 - Object model for:* at page 73).

Parameters:
id id of the new object

3.11.16 PlaySound *[dest,] soundFile*

Sends a sound file (to be played once) to a specific player, all players in a room, all players in the current world. The message frame commonly used to hold the sound clip, so maximum length for the sound file should be the message refresh rate (usually 5 seconds, see *Game Settings*).

Parameters:
dest (Optional) ID of the game object which should receive the sound. It may be either a player, a room, or $WORLD. If omitted, the default is $AGENT.
soundFile (Required) Sound file to be sent.

3.11.17 PlayBackground *[dest,] soundFile [, loop]*

Sends a sound file (to be played in the background - a separate pop-up window is used) to a specific player, all players in a room, all players in the current world.

Parameters:
dest (Optional) ID of the game object which should receive the sound. It may be either a player, a room, or $WORLD. If omitted, the default is $AGENT.
soundFile (Required) Sound file to be sent.

loop (Optional) Set to 1 if you want the sound to be played in loop. Default is 0.

Note: MP3s are played by using the *XSPF Flash player*, capable of playing in progressive download mode.

3.11.18 Print *[dest,] message [, message …]*

Sends a message to player's consoles. May be directed to a specific player, all players in a room, all players in the current world.

Parameters:

dest (Optional) (Optional) ID of the game object which should receive the message. It may be either a player, a room, or $WORLD. If omitted, the default is $AGENT.

message (Required) Message to be displayed. Specifying more messages, they will be used in turn each time the event is triggered.
message can also be an Image object, in which case the image is printed on the screen.

Example (prints a man image on the screen):

```
Print NewImage("uomo.gif",64,100)
```

3.11.19 PrintRight *[dest,] message [, message …]*

Same as Print, but the messages are displayed into a DIV whose ID is right_column. Useful for producing data display in a column placed at the right border of the screen.

3.11.20 Reset

Sends out a broadcast message and resets the game. This is equivalent to re-starting the engine and re-logging in all the players.
No parameters.

3.11.21 RefreshView *[dest]*

Sends a refresh view command to the players' client. Works on a specific player, all players in a room, all players in the current world.

Parameters:

dest (Optional) ID of the game object which should receive the command. It may be either a player, a room, or $WORLD. If omitted, the default is $AGENT.

3.11.22 Return *[retvalue]*

Terminates execution of the current subroutine and returns the specified value. EVENTs returning a non-zero value are considered as "successful" by the game engine which may print out appropriate messages.

Parameters:

retvalue (Optional) Expression whose result is returned.

3.11.23 SaveSetting *key,value*

Stores a specified *value* in the game profiles' database. Value will be associated to the specified *key* and can be retrieved by using the **getSetting** function.

Parameters:

key (Required) String expression. To be used for value retrieval. Please read the section below for special keys, in order to avoid conflicts or weird behaviour.

value (Required) String expression to be stored.

Warning: This instruction involves access to disk, so it is relatively high **time-consuming**. In order to keep a good server performance, consider coupling DimensioneX server with a mySQL database and keep the use of this instruction as low as you can.

3.11.23.1 Special keys

Please take into account that there are keys which have got special meaning or the game engine.

- Keys of the form *something_somethingelse* that is, with an underscore sign ("_") in between, can conflict with game profiles which are saved with under keys of the form: *playerid_propertyvalue*
- The Hall of Fame view searches and uses the following key:
- **hiscoretext** Message to be displayed about the best score ever
- For hi-score management, it is recommended to store value under the following key
- **hiscore** Best performance ever

example of use:

```
            Dim hiscore = 0 + getSetting("hiscore") 'Sum to force considering
number value
            If king.Score > hiscore
                  saveSetting "hiscore",king.Score
                  saveSetting "hiscoretext","Best score is: " + king.Score + " by
<B>" + king.name + "</B>, "+king.Class+ " on " + getTime("MM dd, yyyy")
                  Speak SYS,$WORLD,"The best king is " + king.name + " scoring " +
king.Score
                  Print king,"I am the best king so far!"
            End_If
```

3.11.24 SendCmd *[dest,] command*

Sends a direct command to the players' client, which will be interpeted by the Javascript code contained in the system file named **client.script**. Using this statement is not recommended, as commands for the ***command*** parameter may change in the future. To have a hint on how available ***command***s are managed, look in the **client.script** file of your DimensioneX **system** folder.

Parameters:

dest (Optional) ID of the game object which should receive the command. It may be either a player, a room, or $WORLD. If omitted, the default is $AGENT.

command (Required) String expression to be sent. Possible values are:

- "silence" (Terminates current background music)
- "refresh!scene" (refreshes view)

- "refresh!ctrls" (refreshes controls panel).
- More, custom commands can be sent to the client.
  In this case, specify:
  "**custom:***mycommand!parameter,parameter,…*"
  and include the custom command in the **client.script** file in the javascript function **clientExecuteCmd(what).** Generic, unplanned javascript commands are not accepted by the client.script for security reasons.
  Note: Neither *mycommand* nor *parameter(s)* can contain any semicolon (";"), which is used by the game engine as a separator.

3.11.24.1 Custom Commands

popup!*url,attributes*
Opens a pop-up windows with the specified url and the specified attributes.
Example:

```
SendCmd $AGENT,"custom:popup!https://www.dimensionex.net,width=100,height=100,
resizable=1,status=0,scrollbars=1,directories=0,toolbar=1,menubar=0,left=0,top=0"
```

timed!*commands,interval*
Activates the specified commands after the specified interval (in milliseconds)

3.11.25 SetAdd *set,key,value*

Adds a new element – a (key,value) pair - to the specified *set*.

Note: Sets can also be extended with a simple assignment. Example:
Set(key) = value

See also: SetRemove

3.11.26 SetPanel [*destination,*] *"panelId"*

Sets a specified control panel for a player, a room, or the world, depending on the value of the ***destination*** parameter.

Parameters:

destination	(Optional) Depending on the desired effect, it can be either the ID of a player, the ID of a room, or $WORLD. If omitted, $AGENT is used.
panelId	(Required) String expression representing the panelId. See *2.5.1 - Pre-defined PANELs* to learn about pre-defined panels.

Notes: if SetPanel is targeted $WORLD, a refresh on all player's clients will be triggered. if SetPanel is targeted to a player the refresh in its client will be implicitly done. If the SetPanel is targeted to a room, refresh is not automatically issued.

3.11.27 SetRemove *set,key*

Removes an element to the specified *set*. If used on an ARRAY object instead, the *key* needs to be the index number of the element to be removed.

See also: SetAdd

3.11.28 SendPage (formerly UseView)

Syntax:

```
SendPage destination, pageid, interval
```

Uses a specified **view** (PAGE) for the next output to the specified **destination** player, or as the persistent view for the game if the **destination** is $WORLD. The view is refreshed after the specified **interval**. This is useful to display quick screens to improve action perception (i.e. scenes of a battle, etc) when the destination is a specific player. When the destination is $WORLD the specified view is used as the default world view until changed or reset by another SendPage command sent to $WORLD.

Parameters:

destination (Required) The player receiving the view or $WORLD to set a persistent world view.

pageid (Required) the desired PAGE's ID. The PAGE id must have been declared in the GUI section. Use an empty string ("") to reset a persistent $WORLD view to the standard view. It is not necessary to issue a 2nd SendPage to reset a view sent to a specific player. This happens automatically after the next view output.

interval (Required) Time interval, in seconds, expressing the time after which the view should be reset to the default one. 1,5 is one and half second. By specifying 0 (zero), the view is left on the screen indefinitely.

3.11.29 Speak [*[speaker], dest*] *, message [, message …]*

Speaks to a specified player, all players in a room or all players in current world. If it's a player, the message displays on the message window.
Can also be used also in the simplified form:

Speak *message [, message …]*

Parameters:

speaker (Optional) ID of the character which will send the message. If omitted, the default is $OWNER. If it does not refer to a CHARACTER object, then the system default character is used.

dest (Optional) ID of the game object which should receive the message. It may be either a player, a room, or $WORLD. If omitted, the default is $AGENT.

message (Required) Message to be displayed. Specifying more messages, they will be used in turn each time the event is triggered. At the end of the sequence, the onSpeechFinish event is automatically triggered on the OWNER object
Note: The message "*" can be used to tell the game engine the speech is finished. The message "*" will not appear but a onSpeechFinish event will be triggered instead.

3.12 Functions

3.12.1 Abs(n)

Returns the absolute value of n.

3.12.2 Asc("c")

Returns the ASCII code of the character "c". If a string is specified, only the first character is evaluated.

See also: Chr()

3.12.3 Chr(n)

Returns the character corresponding to the ASCII code n. n must be an integer from 0 to 255. (For the ASCII character table, search on Google with this keywords: "ascii map")

See also: Asc()

3.12.4 Copy(array_or_set)

Returns a copy of the specified object (must be an ARRAY or SET). Please note that this is the most efficient way to get a copy of an ARRAY or SET:

Wrong

```
Dim myArr = NewArray("a,b,c")
Dim myCopy = myArr ' This actually copies the reference to the array
myCopy(2) = "x" ' This will affect both myArr and myCopy!
```

Correct

```
Dim myArr = NewArray("a,b,c")
Dim myCopy = Copy(myArr) ' This actually copies the array, element by element
```

3.12.5 Exists(objectID)

Returns a boolean value which is 1 (true) if any object with the specified *objectID* exists.

3.12.6 ExistScript("scriptID")

Returns a boolean value which is 1 (true) if a script exists, *false* otherwise. It can be either a Sub, Function or EVENT with the specified *scriptID*. Make sure the scriptID is written between quotes.

3.12.7 gameInfo("*variable*")

Returns the value of a few game server's settings.

variable String. Can be either of the following:

- **imagesfolder** Returns the images folder (WORLD.IMAGESFOLDER/ IMAGESFOLDER_PUBLIC/ IMAGESFOLDER_LOCAL according to the current situation)
- **navigator**: Returns the URL of the current game'navigator URL, e.g. https://yourserver.com/dimx/servlet/multiplayer
- **site** Returns the game'swebsite (WORLD.SITE)

Example:

```
Print "Visit this game's website: " + getGameInfo("site")
```

3.12.8 getCharactersIn(object)

Returns a SET of CHARACTERS contained in the specified object.

Example:

```
' Chance for monsters to attack
myset = getCharactersIn($OWNER.container)
Display "Attackable characters for " + $OWNER.name + " are: " + myset
```

3.12.9 getLinksFrom(rooms)

Returns a SET of links starting from the specified room object, or room set.
Rooms can also be a SET or an ARRAY of objects: in this case the result will include links contained in each of the elements (Note: All the set's elements are assumed to be rooms!). Also $WORLD can be specified as a prameter.

3.12.10 getItemsIn(object)

Returns a SET of items contained in the specified object.
Object can also be a SET or an ARRAY of objects: in this case the result will include items contained in each of the elements (Note: All the set's elements are assumed to be objects!)

Example:

```
' Monsters pick up items
Debug $OWNER.name + " is in " + $OWNER.container.name
myset = getItemsIn($OWNER.container)
tobepicked = ""
For Each item In myset
      If Not(item.hidden)
            tobepicked = item
            Display $WORLD,$OWNER.name + " sees " + item.name
      End_If
Next
If tobepicked <> ""
      Move tobepicked,$OWNER
      Display $WORLD,$OWNER.name + " has picked up: " + tobepicked.name
End_If
```

3.12.11 getObject (id)

Searches current world for an object with specified *id*, and returns a reference to it. If *id* is already an object reference, it will be returned unchanged. In practice, getObject guarantees that the result is a valid object reference, or NULL otherwise.

Parameters
id ID of the object you are searching for (or a reference to it)

3.12.12 getObjectsSubtype(container, subtype)

Searches for objects by TYPE by using partial matching. Returns a SET of objects contained in the specified *container* object and of the specified *subtype*.

Parameters
subtype String. Specifies a portion of the TYPE attribute value to search for.

Example:

```
' containsSubtype
' checks whether the specified container contains a specified object type or
subtype
' returns true or false
Function containsSubtype(container,type,recursive)
      If SetLen(getObjectsSubtype(container,type)) > 0
            Return true
      End_If
      Dim res = False
      If recursive ' another chance: look inside objects
            Dim setobjects = getItemsIn(container)
            Dim o
            For Each o In setobjects
                  res = res Or containsSubtype(o,type,true)
            Next
            Return res
      End_If
      Return res
End_Function
```

3.12.13 getObjectsType(container, type)

Searches for objects by TYPE. Returns a SET of objects contained in the specified *container* object and of the specified *type*.

Parameters
type String. Specifies the TYPE attribute value to search for.

Example:

```
        ' Gets all the swords at the ground
        myset = getObjectsType($AGENT.container,"sword")
        If SetLen(myset) > 0 ' If any…
            Display "Swords available on the floor! I get one."
```

```
            Move myset(1),$AGENT
        End_If
```

Returns a SET of items matching the specified type.

Note: If the object uses a composit type.subtype, the function will return all those object with a successful, partial match on the main type. That is, items with sword.iron, sword.short, sword.long will be returned. If you need a partial match over the subtype part, then use **getObjectsSubtype**.

3.12.14 getPlayerProperties(user_name)

Searches the players' profile DB and returns a SET of properties indexed by the names of the various properties.

Example:

```
'Searches for specified user NAME and parameter
'Returns a string value
Function LookupProfileDB(user_name,parameter)
      Dim props = getPlayerProperties(user_name)
      Return props(parameter)
End_Function

Dim userlevel = LookupProfileDB("Cris","Level")
```

3.12.15 getPlayersIn(object)

Returns a SET of players contained in the specified object.

Example:

```
        ' Chance for monsters to attack
        myset = getPlayersIn($OWNER.container)
        Display "Attackable players for " + $OWNER.name + " are: " + myset
```

3.12.16 getRooms ([setExclusions])

Returns a SET containing all current world's ROOMS.

Parameters:
setExclusions (Optional) SET of rooms to be excluded from the result (if any).

Example:

```
        myset = getRooms(Start)
        'Display "All rooms except initial one are: " + myset
        destination = RndSet(myset)
        If RndInt(3)=1 And destination <> ""
            'Once on a 3 times and if valid destination
            Move $OWNER,destination
            Speak $OWNER,$WORLD,"Moving to " + destination.name
        End_If
```

3.12.17 getRoomsFrom(room)

Returns a SET of ROOMS which can be accessed from the specified *room*.

Example:

```
myset = getRoomsFrom($OWNER.container)
'Display "Accessible rooms for " + $OWNER.name + " are: " + myset
destination = RndSet(myset)
If RndInt(3)=1 And destination <> ""
    'Once on a 3 times and if valid destination
    Move $OWNER,destination
    Speak $OWNER,$WORLD,"Moving to " + destination.name
End_If
```

3.12.18 getSetting(*key, [default]*)

Retrieves a setting from the game profiles database. Returns the value associated to the specified *key*. See also the instruction: *SaveSetting key,value*.

Parameters:

key	Key to be seached in the database
default	Default value to be returned if the key cannot be found

Warning: This function involves access to disk, so it is relatively high **time-consuming**. In order to keep a good server performance, consider coupling DimensioneX server with a mySQL database and keep the use of this function as low as you can.

3.12.19 getTime("*format*")

Returns the server's system date/time, depending on the "format" string.

Format must be a string including spaces, punctuation and any of the following elements:

HH	Hour
mm	Minutes
ss	Seconds
MM	Month
dd	Day of month
yyyy	Year

The above is only a partial list, since this function replicates the behaviour of the Java class SimpleDateFormat. For a complete explanation of possible values and examples see: http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html

Example:

```
' Display current time
Display SYS,$WORLD,"Time is now: " + getTime("HH") + ":" + getTime("mm")
```

3.12.20 HttpFetch(*url*)

Fetches a page or file from the internet and returns it as a string.

Note: By using this function you GREATLY slow down the game! Use it with parsimony.

Parameters
url URL starting with https:// or http://

Example:

```
' Fetches google's homepage
Dim txt = HttpFetch("https://www.google.com")
```

3.12.21 InStr(*haystack,needle [,startpos]*)

Searches *needle* in *haystack* and returns the first position at which *needle* was found. The search is NOT case sensitive. The first position in the string is 1, If *needle* is not contained in *haystack* the result will be zero.

Parameters
startpos (Optional) Positive integer. Specifies the position to start searching from. The first position is 1.

3.12.22 InStrCount(*haystack,needle*)

Searches *needle* in *haystack* and returns the number of occurences of *needle*. The search is NOT case sensitive. If *needle* is not contained in *haystack* the result will be zero.

Note: If you want to test whether or not needle is contained in haystack, **InStr** function will work faster than **InStrPos**, because **InStr** will stop searching as soon as the first occurrence is found.

3.12.23 Int(n)

Returns the integer part of the argument n.
Note: Int does not rounds up, just truncates the number. To round up the number, use **Round**:

```
nRounded = Int (n + 0.5)
```

3.12.24 IsCharacter( )

Returns **true** when the argument is a CHARACTER. Please note that players are also characters.

3.12.25 IsPlayer( )

Returns **true** when the argument is a player. Cannot be used on a non-existent object.

3.12.26 IsRoom( )

Returns **true** when the argument is a ROOM. Cannot be used on a non-existent object.

3.12.27 LCase(*string*)

Returns the specified *string* transformed to lowercase.

Example:

```
Display LCase("mAsTeRpLaN")
```

Displays: masterplan

3.12.28 Left(*string,nchars*)

Returns the first *nchars* of the specified *string*.

3.12.29 Len(*string*)

Returns the length, in characters, of the specified *string*.

3.12.30 Log(*number*)

Returns the natural logarithm of the specified *number*.

3.12.31 MainType(*object*)

Returns the specified *object*'s main type.

```
Dim knife = NewItem(hellfire,"bloody knife","It is a
knife",NewImage("uw2/knife2.gif",64,64),"type=weapon.knife,pickable,Power=1,Value=0
,sound=sword1.wav,icon=uw2/knife.gif,showmode=2")
Print MainType(knife)
```

Displays: weapon

3.12.32 Mid(*string,start,nchars*)

Returns the *nchars* starting from position *start* in the specified *string*. Positions are numbered starting from 1 (for the first character in the string)

Example:

```
Display Mid("abcde",2,1)
```

Displays: b

3.12.33 NewArray(*[string]*)

Creates a new ARRAY object by splitting a string, and returns it as the result.

Parameters

string (Optional) Specifies the array contents, in the form: "value1, value2, …".

3.12.34 NewCharacter(*room,name,description,image,attributes*)

Creates a new computer-controlled CHARACTER with the specified *name, description,image* and *attributes*. The new character is placed in the specified *room*. Returns a reference to the new character.

Notes:
- *attributes* must be specified as a string, they must be a comma-separated list of pairs as for the CHARACTER ATTRS tag specifications.

Example:

```
      new = NewCharacter(Start,"a vampire","He is named
Dracula",NewImage("vampire.gif",95,100),"Dexterity=6,Strength=5")
```

3.12.35 NewImage(*url,width,height*)

Creates a new IMAGE with the specified *url, width* and *height*, then returns it.

3.12.36 NewLink(*from,to,orientation,bidirectional,name,description,image,attributes*)

Creates a new LINK between rooms from and to, with specified *orientation*, *name, description,image* and *attributes*. The new link may be or not *bidirectional*, as specified. Returns a reference to the newly created link.

Parameters

from	ROOM object: where the link should start from.
to	ROOM object: where the link should arrive to.
orientation	String: Establishes the orientation of the link. It must be one of the following: N/S/E/W/U/D.
bidirectional	Boolean. Specifies whether or not the link will be bidirectional. True or 1 for yes.
name	String. What the user will read. Specify "" or null for automatic setting based on orientation ("East,West,…").
descripton	String. What the user will read when using the LOOK command.
image	(Optional) IMAGE object. Specify **null** to set it later.
attributes	String. What you would normally specify as attrlist. Also icon can go in here.

3.12.37 NewSet(*[string]*)

Creates a new SET.

Parameters

string (Optional) Specifies the set contents, in the form: "key1=value1,key2=value2". If this parameter is missing, an empty, sorted set is produced.

Notes

- NewSet always produces a sorted SET. If you want to produce an unsorted set, here's a tip: Use getItemsIn(object) where object is an object that does not contain anything, then extend it.
- For the *string* parameter, if you need to use "," and "=" characters in values, they can be specified via the escape sequences: "!1!" and "!2!".

3.12.38 NewItem(*container,name,descr,image,attributes*)

Creates a new dynamic ITEM with the specified *name,description,image* and *attributes*. The new item is placed in the specified *container*. Returns a reference to the new ITEM.

Notes:
- *attributes* must be specified as a string, they must be a comma-separated list of pairs as for the CHARACTER ATTRS tag specifications.
- If the target container han not enough free space to hold the newly-created item (attributes *container*.capacity and item.volume) then the item is placed in *container*'s container.
- The placement of the newly created item into container **will not trigger** movement events (whenPicked, onReceiveItem). This is because **NewItem** is often used inside movement events and hence it may cause infinite recursion deadlocking the game. If you need the movement events to be triggered then you should create the object in the current room (as in the following example), and then **Move** the object so that events are normally triggered.

Example:

```
      new = NewItem($AGENT.container,"an object","What a nice
object",NewImage("object.gif",95,100),"volume=2,pickable=1")
      Move new,$AGENT
```

3.12.39 NOT( )

Negation – remember the argument has to be put between brackets!

3.12.40 PanelHtml("*panel*")

Outputs the specified *panel* in HTML format. Use this function to produce context-sensitive menus.

Example:

```
GUI
PANEL pronald
      BUTTON dothis, "Do this: ", "Comando", doCommand
      DROPDOWN ronaldops, "getChoices(ronald)", 0

END_GUI

SCRIPTS

EVENT ronald.onLook
      Print PanelHtml("pronald")
      Speak "I have learnt to listen to the words of the trees... They taught me
their secrets.","Tell me what you want me to do."
End_EVENT

END_SCRIPTS
```

3.12.41 Replace(*haystack,needle,replacement*)

Searches a string (*haystack*) for finding another string (*needle*) and replace it with a string (*replacement*), returning the result. The search is case-sensitive.

3.12.42 Right(*string,nchars*)

Returns the last *nchars* of the specified *string*.

3.12.43 RndInt(*N*)

The very bread and butter of the games. Return an integer number between 1 and N.
Beware: N must be a positive number

3.12.44 Rnd()

Returns a floating-point number between 0 and 1.

3.12.45 RndSet(*aSet*)

Returns a random element of the specified set or array.
Useful for placing objects and characters in random rooms.

3.12.46 Round(*number,int*)

Returns the specified *number* rounded to the specified *int* decimals.
Example:

```
Display Round(3.509,2)
```

Displays

```
3.51
```

3.12.47 SetContainsKey(*set,key*)

Returns *true* when the specified *set* contains the specified *key*.
Example:

```
…

SETS
SET   setCovered  chapel,bar, bedroom
END_SETS


Sub checkLight(person)
' Die for vampires when in sunlight
' Input: target character
      If Not(SetContainsKey(setCovered,person.container.id))
            ' Not in bedroom nor in covered places
            Display person,"AAARGHHH!!! Sunlight is killing me!!"
            Kill target
      End_If
```

In the above example, if *person* is NOT in one of the rooms listed in setCovered, a call to checkLight(person) will make person to see a message and die.

Note: Do not use this function on ARRAYs as the check is performed on the keys, not the values.

3.12.48 SetIndexOf(*setarray,value*)

Returns the numeric index of *value* inside *setarray* (it can be either a SET or an ARRAY).
If value is found, the result is a number N where: 1 <= N <= setLen(setarray)
If value is not found, it returns 0 (false).
The search is *case-insensitive*.

Somewhat similar to **SetContainsKey**, checks in values instead of keys.

3.12.49 SetKey(*set,n*)

Returns the key at position *n* of the specified *set*.

3.12.50 SetKeys(*set*)

Returns an ARRAY containing the keys of the specified *set*.

3.12.51 SetLen(*set*)

Returns the number of elements contained in the specified *set*. If the argument is not a set, -1 is returned.

Parameters:
set can be either a SET or an ARRAY

Example:

```
possibilities = Split("pick/attack/go/drop/bite","/")
Display $WORLD,"You have: " + SetLen(possibilities) + " possibilities"
```

produces:

```
You have 5 possibilities
```

3.12.52 Split(*valuestring,separatorstring*)

Splits the specified *valuestring* according to the specified *separator*. Produces an ARRAY whose values are extracted from the string.

Example:

```
possibilities = Split("pick/attack/go/drop/bite","/")
Display $WORLD,possibilities
```

Constructs the following set: (pick,attack,go,drop,bite)

3.12.53 Sqr(x)

Returns the square root of the argument *x*.

3.12.54 UCase(*string*)

Returns the specified *string* transformed to UPPERCASE.

Example:

```
Display UCase("mAsTeRpLaN")
```

Displays: MASTERPLAN

3.13 Methods

Methods are similar to functions and instructions, but they are very closely related to an object. In other terms, a method is *what the object is capable to do*.

All methods must be called in this form:

object.method()

If you don't expect a result value, the Call instruction must be used.

Example:

```
Print $AGENT.getCookie("myname")
```

Or

```
Call $AGENT.saveCookie("myname",$AGENT.name)
```

4 SKINS - Customizing the user interface

A SKIN is a combination of HTML settings and image files to be used for the graphic parts of the players' interface.
You can define your own custom skin for games by writing a text file with .DXS extension according to the DXS syntax. After soing so, your skin definition (.DXS file) and media files can be easily re-used in different games and servers. On the DimensioneX web site we publish skins produced by users, and credit the designers.
The best approach for writing a custom skin is probably to start by modifying one of the skins you will find in the DimensioneX kit.

4.1 Placement of SKIN files

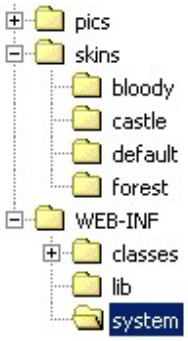
a SKIN is composed by:
1. a DXS file defining the skin. This file should have a .DXS extension (recommended) but if your web provider doesn't allow them, you can also use a .TXT extension.
2. a number of media files (GIF, JPG or PNG and WAV) referenced by the DXS file.

The placement policy of skin files has been changed since version 6.4.4.
For the placement of files, please follow this guide carefully.

Supposing your skin is described in the file "**myskin.dxs**":
1. The myskin.dxs or myskin.txt file must be contained in its folder with the same name as the file, hence in our example you have to create a folder named **myskin** inside the **skins** folder.
2. If your game's images are hosted elsewhere, say, on a webserver that is different than the one hosting the DimensioneX engine, the skin's folder must be copied also on that server, too. The skin folder on the media server must be parallel to the IMAGESFOLDER you are using (i.e. they must be contained both in the same folder)

Example: If your game's image files are in:
https://whateverserver /dimx/pics
then the game engine will assume that the skin media files are stored in:
https://whateverserver/dimx/skins/*myskin*

The picture below shows part of the DimensioneX subtree. You can see the game's pictures folder **pics** and below the **skins** folder which contains the folders for each skin in use: **default**, **bloody**, **castle** and **forest**.



4.2 DXS syntax

Example: this is the definition for a skin mamed Forest, see file **forest.dxs** in your DimensioneX system folder:

```
SKIN    forest
        ' Version 1.0 - by Cris
        NAME    Forest
        PICPLAYER       avatar.gif
        PICSPACER       blank.gif
        PICFADER        fadewhite.gif
        ICOPLAYER       player.gif
        ICOCHARACTER     character.gif
        ICOITEM         item.gif
        ICOWAY          way.gif
        ICOWAYN         north.gif
        ICOWAYS         south.gif
        ICOWAYE         right.gif
        ICOWAYW         left.gif
        ICOWAYU         up.gif
        ICOWAYD         down.gif
        ICOROTL         icorotl.gif
        ICOROTR         icorotr.gif
        ICOREV          reverse.gif

        SNDNEWMSG       notify4.wav

        BODYBGCOLOR             #206020
        BODYBACKGROUND
        BODYSTYLE     {font-size:10pt;font-family:Courier New;color:#00FFFF}
        SYSMSGSTYLE     {font-size:10pt;font-weight:bold;font-family:Courier New;color:#FF1111}
        LINKSTYLE     {font-size:10pt;font-family: Lucida Sans;color:#005000}
        ALINKSTYLE     {font-size:10pt;font-family: Lucida Sans;color:#FF0000;text-decoration: none}
        ICONLABELSTYLE     {font-size: 8pt; font-family: Lucida Sans;color:#DDDDDD}
        PANELBGCOLOR     SILVER
        PANELBACKGROUND     vine-001.jpg
        PANELSTYLE     {font-size:10pt;font-family:Lucida Sans;color:#FFFFFF}
        BUTTONSTYLE     {background-color: #00CC00; color: black; cursor: hand; font-family: Tahoma;
font-size: 8pt; height: 20px; border-top: 2px outset #00FF00; border-left: 2px outset #00EE00;}
        TITLESTYLE     {font-size:10pt;font-family:Lucida Sans;font-weight:bold;color:#FFFF20;background-
color:#009000}
        LIST1BGCOLOR     #004000
        LIST2BGCOLOR     #316131
        LISTSTYLE     {font-size:8pt;font-family:Courier New;color:#DDDDDD}
        STYLESHEET      stylesheet.css
END_SKIN
```

An explanation of the fields follows. Only the first two are mandatory, all others are optional. If they are not specified, the default skin values are used.
Lines beginning with an ' are treated as comments.

SKIN attribute	Meaning
ID	Identifier (mandatory). **Special value:** "default" tells the system this is going to be the new default skin.
NAME	Skin's name (mandatory). The player sees this in the drop-down list at the login panel.
SKINFOLDER	(Optional – usually omitted) It is an URL with final slash, and specifies in which folder are kept images used by this skin.

	If not specified, a default folder named **dimx/skins/ID** is used. You may specify an absolute URL such as: https://www.dimensionex.net/pics/skins/forest/ or a relative URL referred to the local host, such as: /dimx/skins/forest/ **Pay attention:** the final slash is required.
PICPLAYER	Image which represents other players. By default is assumed to have a size of 64 x 100 pixels, so don't use images with very different proportions or sizes.
PICSPACER	Image which the system uses to create spacings. It is a single transparent pixel. Sometimes, for debugging purposes, it may be useful to use an opaque image.
PICFADER	Image used to obscure the background scene when looking at something. Usually a half-transparent GIF image is used.
ICOPLAYER	Icon representing other players.
ICOCHARACTER	Default icon for computer-controlled CHARACTERS.
ICOITEM	Default icon for ITEMS.
ICOWAY	Default icon for WAYs with no DIRECTION assigned.
ICOWAYN/S/E/W/U/D	Default icon for WAYs for which a DIRECTION has been specified and corresponds to N/S/E/W/U/D
ICOROTL/R	Icon or button used to rotate the player towards left/right in rooms having more than one image.
ICOREV	Icon or button used to reverse the player in rooms having more than one image.
SNDNEWMSG	Audio clip to execute when the player receives a message.
BODYBGCOLOR	Background color
BODYBACKGROUND	Background image. Null string ("") for no-one.
BODYSTYLE	Font style for normal text. Null string ("") for browser default.
SYSMSGSTYLE	Font style for system messages (usually: game engine errors). Null string ("") for browser default.
LINKSTYLE	Font style for hypertext links.
ALINKSTYLE	The same, for activated links (useful for rollover effects). Warning, it seems Netscape does not support them.
ICONLABELSTYLE	Font style for icon labels.
PANELBGCOLOR	Background color for command panel.
PANELBACKGROUND	Background image for command panel. Null string ("") for no-one.
PANELSTYLE	Font style for labels and buttons in the command panel.
BUTTONSTYLE	Style to be applied to command buttons. In the background specification you can use the following **special values**: - $ICON is automatically replaced by the buttons' custom icon's url (i.e. specify the ICON tag in the panel definition), if present. - $SKINICON is automatically replaced by: (the skin's folder)/pan(command id).gif

	this is the quickest way to produce button icons but you have to ensure that there are icons in the skin's folder with the correct naming as specified above. (for an example, see out Forest skin)
TITLESTYLE	Font style for the title bar. Specify also the background if you want to change it. **Note:** You can specify $SKINFOLDER to reference the skin's media folder.
LIST1BGCOLOR	Background color for odd-numbered rows in the messages windows.
LIST2BGCOLOR	The same, but for even-numbered rows.
LISTSTYLE	Font style for the messages window.
CTRLBANNER	Contains an IMAGE tag to be shown on top of the command panel. It may be the game logo…
STYLESHEET	Name of the file (must be included in the SKIN images folder) definiing additional styles (see section below). It must be standard stylesheet in CSS syntax.

All "Styles" have to be defines as foreseen by the w3c CSS stylesheets syntax. For more details you may refer to the following site:

http://www.w3.org/TR/REC-CSS2/cover.html#minitoc

A good example of skin using styles is the skin named "Bloody Hell", used in the *Underworld* game. Download it from the DimensioneX site under the Resources section.

4.2.1 STYLESHEET - Additional styles

Some elements of the user interface are directly linked to CSS classes and IDs. These classes and IDs can be specified via a standard CSS file that can be specified by means of the STYLESHEET skin tag, and that must be located in the SKIN's folder.

This is a list of the classes and IDs that DimensioneX uses but you can add your own.

identifier	Type	What is it for
#navpad	ID	Style for the navigation pad (it is a TABLE)

In the future, most elements of DimensioneX UI will be customised according to these styles.

4.2.2 Packing a skin for sending it to other users

Skin have been designed in order to be easily exchanged between developers.
To pack a skin:

- Copy the skin.dxs file into your skin's folder
- Pack into a ZIP file the whole skin folder. In order to do this using WinZip, you simply right-click your skin's folder and choose from the pop-up menu: Add to skinname.zip where "skinname" would actually be your skin's folder name. By doing this, you should have preserved the original folder in the ZIP.

- Send the ZIP to your contact

4.2.3 _net skin versions

Skins bundled with the DimX package are provided with a second version whose name includes the "_net" suffix. These skins can be quickly used on whatever game you like, they will force the browser to load the images to the same, pre-defined, remote server.

Even though these skins are good to get things started fast, it is always safer for production installations to have a copy of all skins hosted on your own server, so that you are not bound to any specific third-party service.

4.2.4 Troubleshooting SKINs

- If the images don't show up, go read *4.1 - Placement of SKIN files* and correct the placement of skin media files.
- If this don't solve your problem, then probably the SKINFOLDER tag has an incorrect value. Try opening the skin.dxs file and removing the SKINFOLDER tag.
- Always remember that the debug.log files will contain messages that can help you diagnose skin loading problems.

5 About DimensioneX Client

5.1 How it works

The DimensioneX client is rendered through the browser, so that a player does not have to download anything.
The HTML to be visualised by the browser, however is generated by the game engine by using some templates which are:

- The **client.script** file (for the Javascript code)
- The chosen **.client** file (which is actually near-HTML code)

The communication between the HTML client and the game engine is done through a simple protocol based on the standard HTTP POST and GET methods.
It is relatively easy to write customised custom clients for DimensioneX provided they are capable of displaying HTML, running Javascript, handling CSS/Styles and managing "cookies".

5.2 Calling the dimensioneX client

The client is obtained by calling the DimensioneX main servlet, which is:

```
http://server:port/dimx/servlet/cleoni.adv.multiplayer?game=N
```

(also https:// will work)
where N represents the server slot (default value is 1)

or simply (in web servers with a proper servlet mapping defined, such as for Tomcat)

```
http://server:port/dimx/servlet/multiplayer?game=1
```

The game engine can host unlimited games, each one associated to a different slot:

```
http://server:port/dimx/servlet/multiplayer?game=4096
```

actually runs the game assciated to server slot number 4096 (a config file named worldnav4096.properties is assumed to be present in the system folder).

5.2.1 Selecting the proper screen size

DimensioneX can properly handle different screen sizes, provided that the game programmer has correctly set the screen size for which the game's default scene size was designed.
Since version 7.0.3, it is possible to handle cellphone browsers by using specially designed clients.

To automatically call up the client for a specified screen mode, the following syntax is used:

```
https://server:port/dimx/servlet/cleoni.adv.multiplayer?game=1&width=WWWW&height=HHHH[&client=clientfile][&locked=1]
```

Where WWWW and HHHH are replaced by the desired screen dimensions.

locked=1
can be added to ensure that the user cannot change the pre-selected resolution (ideal for page-embedded games)
client=clientfile
tells the game engine to use a specific client file for the connection (clientfile.client will be searched) in the dimx/system directory.

Example:

```
https://play.dimensionex.net/dimx/servlet/multiplayer?game=1&width=1024&height=768
```

Calls up the client for a resolution of 1024x768.

Here is a Javascript to determine resolution and launch a game window it in your game's page:

```
<script type="text/javascript" language="JavaScript">
function launch(url) {
      // Startup code for the client
      var w = screen.width - 10;
      var h = screen.height - 100;
      var attrs =
"resizable=1,status=1,scrollbars=1,directories=0,toolbar=1,menubar=0,left=0,top=0,w
idth="+w+",height="+h;
      //alert("'" + attrs + "'");
      game = window.open(url +
"&width="+screen.width+"&height="+screen.height,'game',"'" + attrs + "'");
}

</script>



<a
href="javascript:launch('http://play.dimensionex.net/dimx/servlet/multiplayer?game=
1');">Click here to play</A>
```

Notes

- If the resolution is not specified while calling the client, then a screen mode selection menu is automatically added to the game logon controls.
- If the pre-selected resolution is not supported by the available game resolutions, 640x480 (VGA) gets automatically pre-selected.

5.3 Who's online - The players view

Usually, in a game's web site, it is useful to show how many players are connected to the online game. This can be done easily by means of an IFRAME, that is inserting into a web page an HTML code of the following form:

```
<IFRAME HEIGHT=26 WIDTH=100 SCROLLBARS="0" SCROLLING="no"
SRC="https://www.myserver.com/dimx/servlet/multiplayer?view=players&game=slot"></IF
RAME>
```

Where:

slot Is the slot number hosting the game

If the game is multi-area, then use this syntax:

```
<IFRAME HEIGHT=26 WIDTH=100 SCROLLBARS="0" SCROLLING="no"
SRC="https://www.myserver.com/dimx/servlet/multiplayer?view=players&game=slot&clust
er=clustername"></IFRAME>
```

Where:

clustername Is the cluster name the game belongs to
slot Is the slot number hosting any of the worlds of the cluster

5.3.1 Showing players' positions

If you would like to see where the players are, then slightly modify the HTML code like this:

```
<IFRAME HEIGHT=200 WIDTH=400
SRC="https://www.myserver.com/dimx/servlet/multiplayer?view=players&format=extended
&game=slot&cluster=clustername"></IFRAME>
```

Here we added the **format=extended** parameter, which activates the extended view, showing all players' position. Remember to remove the **cluster=** part if your game is not multi-area.