AssetCenter™

Version 3.0

WebKit User Guide

April 22, 1999

ITEM00449

Peregrine SYSTEMS®

AssetCenter data integrity

AssetCenter is extremely rich in functionality. This richness relies on a complex database structure: The database contains a large number of tables, fields, links and indexes; certain intermediary tables are not displayed by the graphical interface; certain links, fields and indexes are automatically created, deleted or modified by the software.

Only the interfaces designed for AssetCenter (graphical interface, APIs, import program, WEB interface and gateways) are capable of modifying the database with respect to its integrity. **You must never modify the structure and/or the contents of the database by any means other than those intended for use with the software**; such modifications are highly likely to corrupt the database and bring about symptoms such as involuntary loss or modification of data or links, creation of "ghost" links or records, serious error messages, etc. Alterations to the database resulting from manipulations of this type void the guarantee and technical support provided by Peregrine Systems.

Environments supported by AssetCenter

The list of environments supported by AssetCenter can be found in the manual entitled "Installation and Upgrade Guide". Using AssetCenter in environments other than those for which it is intended is done at the risk of the owner. Alterations made to the database resulting from using AssetCenter in environments other than those for which it is intended void the guarantee and technical support provided by Peregrine Systems.

Foreword

This manual provides detailed explanations on the use of AssetCenter WebKit.

AssetCenter WebKit is designed for enterprises who are implementing AssetCenter Web and want to customize the user pages displayed, either by modifying existing pages or by creating new ones.

Certain skills are required to use AssetCenter WebKit:

- an understanding of Web mechanisms and architecture,
- knowledge of HTML and a web page design tool,
- BASIC programming experience to create new pages,
- knowledge of the structure of the AssetCenter database.

This document will not allow users with no knowledge of these areas to use AssetCenter WebKit.

This documentation is a complement to the "AssetCenter Web User's Guide". It may itself be complemented by the "AssetCenter: Programmer's Reference" manual.

The BASIC used in AssetCenter Web is a sub-set of BASIC developed by CypressInc. and is similar to "Visual Basic for ApplicationsTM". Only certain functions of "Visual Basic for ApplicationsTM" are supported.

Contact Peregrine Systems

World Headquarters

Peregrine Systems, Inc.
3611 Valley Centre Drive
San Diego, CA 92130
USA
Tel: +1 619 481 5000 or 800 638 5231
Fax: +1 619 481 1751
Web: http://www.peregrine.com

Customer support:
Tel: +1 619 794 7402 or 800 960 9998
Fax: +1 619 794 6028
Web: http://support.peregrine.com
E-mail: support@peregrine.com
Open Monday to Friday 5:00 AM to 5:30 PM (PST)

France

Peregrine Systems
Tour Franklin - La Défense 8
92042 Paris - La Défense Cedex
France
Tel: +33 (0)1 47 73 11 11
Fax: +33 (0)1 47 73 11 12

Customer support:
Tel: +33 (0) 800 505 100
Fax: +33 (0)1 47 73 11 61
E-mail: frsupport@peregrine.fr
Open Monday to Friday 8:00 AM to 6:00 PM (local time)

Germany

Peregrine Systems GmbH
Bürohaus ATRICOM
Lyoner Strasse 15

60528 Frankfurt
Germany
Tel: +49 (0)69 66 80 260
Fax: +49 (0)69 66 80 2626

Customer support:
Tel: +49 (0)69 66 80 260
Fax: +49 (0)69 66 80 2626
E-mail: psc@peregrine.de
Open Monday to Friday 8:00 AM to 5:00 PM (local time)

United Kingdom

Peregrine Systems, Ltd.
Ambassador House
Paradise Road
Richmond
Surrey TW9 1SQ
UK
Tel: +44 (0)181 332 9666
Fax: +44 (0)181 332 9533

Technical Support E-mail: uksupport@peregrine.com

Customer support:
Tel: +44 (0)181 334 5890
E-mail: uksupport@peregrine.com
Open Monday to Friday 8:00 AM to 6:00 PM (local time)

Denmark

Peregrine Systems AS
Naverland 2, 12th fl.
2600 Glostrup
Denmark
Tel: +45 4346 7676
Fax: +45 4346 7677

Customer support:
Tel: +45 7731 7776
Fax: +45 4346 7677
E-mail: nordic@peregrine.com
Open Monday to Friday 8:30 AM to 4:00 PM (local time)

The Netherlands, Belgium and Luxembourg

Peregrine Systems BV
Botnische Golf 9a
Postbus 244
3440 AE Woerden
The Netherlands
Tel: +31 348 43 7070
Fax: +31 348 43 7080

Customer support:
Tel: 0800 0230 889 (The Netherlands only)
Fax: +31 348 43 7080
E-mail: benelux.support@peregrine.com
Open Monday to Friday 8:00 AM to 6:00 PM (local time)

Singapore

Peregrine Systems Pte.Ltd
#03-16
CINTECH III
77 Science Park Drive
Singapore Science Park
118256
Singapore
Tel: +65 778 5505
Fax: +65 777 3033

Italy

Peregrine Systems, S.r.l.
Via Monte di Pietà, 21
I-20121 Milano
Italy
Tel: +39 (02) 86 33 72 30
Fax: +39 (02) 86 33 77 20

Customer support:
Tel: +39 (02) 86337230
Fax: +39 (02) 86337400

Sweden

Peregrine Systems AB
Frösundaviks Allé 15, 4th floor
S-169 70 Solna
Sweden
Tel: +46 8 655 36 04
Fax: +46 8 655 26 10

Customer support:
Tel: +45 7731 7776
Fax: +45 4346 7677
E-mail: nordic@peregrine.com
Open Monday to Friday 8:30 AM to 4:30 PM (local time)

Japan

Peregrine Systems K.K.
Level 32, Shinjuku Nomura Building
1-26-2 Nishi-shinjuku, Shinjuku-ku
Tokyo 163-0532
Japan

Tel: +81 (3) 5322-1350
Fax: +81 (3) 5322-1352

Asia and Pacific

Customer support (based in USA):
Tel: +1 619 794 7402 or 800 960 9998
Fax: +1 619 794 6028
E-mail: apsupport@peregrine.com
Open Monday to Friday 5:00 AM to 5:30 PM (PST)

Conventions

The following notation is used for commands:

[ ]	Square brackets denote an optional parameter. Do not type them in your command. Exception: In BASIC scripts, square brackets are used to denote the data access path and must be included in the script: **`[Link.Link.Field]`**
< >	Brackets denote a parameter in plain language. Do not type them. Substitute the text with the appropriate information.
{ }	Curly brackets denote a series of parameters. Only one of these parameters may be used. Do not type these curly brackets in your command.
|	A pipe is used to separate a series of parameters contained within curly brackets.
*	An asterisk added to the right of square brackets means that the formula shown can be repeated several times.

The following text formats have given meanings:

`Fixed width characters`	DOS command.
`Example`	Example of code or command.
...	Code or command omitted.
"Object name"	The names of fields, tabs, menus and files are shown within double quotes.
Note	Important note.

Send us your comments

We want to deliver the most accurate documentation possible.

Any comments would be greatly appreciated.

Send any remarks to documentation@peregrine.com.

Table of contents

Chapter 1 - Overview

Web technologies

The Web consists of a collection of software, protocols and standards. Hypertext links make it easy to navigate through the Web and access all kinds of documents.

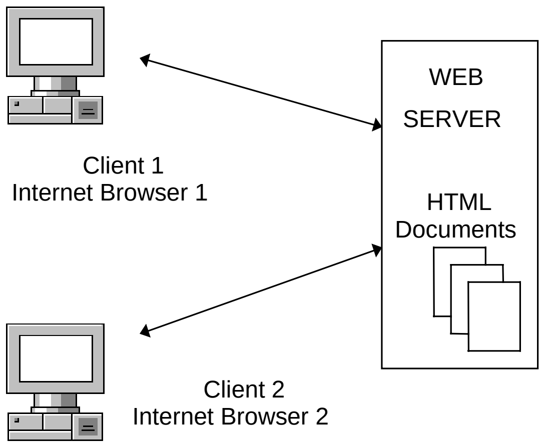
AssetCenter Web allows any user in the enterprise with a web browser to access the data managed in AssetCenter.

Web architecture

Client/Server structure

The Web is built around a client/server architecture:

- Clients use an Internet browser such as Microsoft Internet Explorer or Netscape Navigator to send requests for documents to the web server.
- The server provides HTML documents, which are generally static documents stored on the server. They may refer to other documents. This structure allows users to access documents of interest by simply clicking on the appropriate links.



Typical Web architecture

Standards

Two standards are at the basis of the Web:

- HTTP ("HyperText Transport Protocol"), for data transport.
- HTML ("HyperText Markup Language"), for document presentation.

HTTP

This is the transport layer protocol of the Web.

HTTP is a text protocol that defines a list of commands for requesting documents from or submitting documents to a Web server. The browser handles this protocol transparently for the user.

Note: AssetCenter Web is compatible with version 1.0 of the HTTP protocol.

HTML

This is a presentation standard for hypertext documents. It specifies a set of keywords for structuring a document.

HTML documents are text documents containing information and presentation instructions. These instructions define:

- styles,

- links,
- forms,
- images,
- etc.

The presentation instructions are defined by markers, called “tags”. Tags are generally used in pairs (an opening tag and a closing tag). They delimit the area to which the presentation is applied.

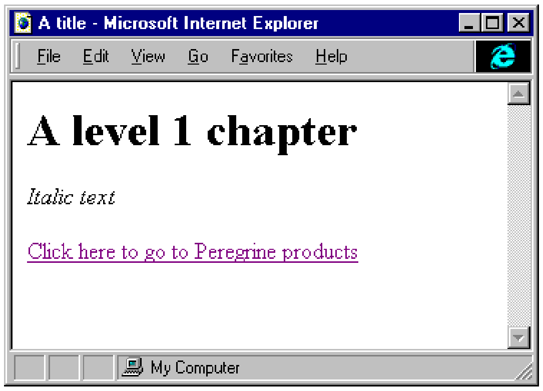
The browser renders the HTML text on the screen.

Note: AssetCenter Web generates HTML text compatible with version 3.2.

Example of a simple HTML text:

```
<TITLE> A title </TITLE>
<H1> A level 1 chapter </H1>
<EM> Italic text </EM> <BR> <BR>
<A HREF="http://www.peregrine.com"> Click here to go to Peregrine
products </A>
```

Here is the resulting page:


A title - Microsoft Internet Explorer
File Edit View Go Favorites Help
My Computer

A level 1 chapter

Italic text

Click here to go to Peregrine products

Warning: An exhaustive list of all HTML tags is beyond the scope of this document. For further information on HTML, we suggest you refer to the extensive reference materials available on this subject. You can also visit The web site of the W3 consortium at "www.w3.org".

Limitations of Web systems

In traditional Web systems:

- Web users are anonymous.
- Web servers do not maintain status information. They cannot retain historical records.
- Only “static” data is sent. The data is contained in HTML documents stored in a folder on the server.

For the AssetCenter application, however:

- Users must be identified.
- Session information must be stored so that users can work in a specific context defined by their user profile.
- Data must reflect the state of the AssetCenter database in real time.

AssetCenter Web

Objectives

AssetCenter Web was designed to:

- Access the AssetCenter database over a network, using an ordinary Web browser.
- Execute simple read and write operations on data in the AssetCenter database in real time.
- Allow for Web page customization. Users can design their own pages or modify existing ones using a simple text editor or an HTML editor.

Technological choices

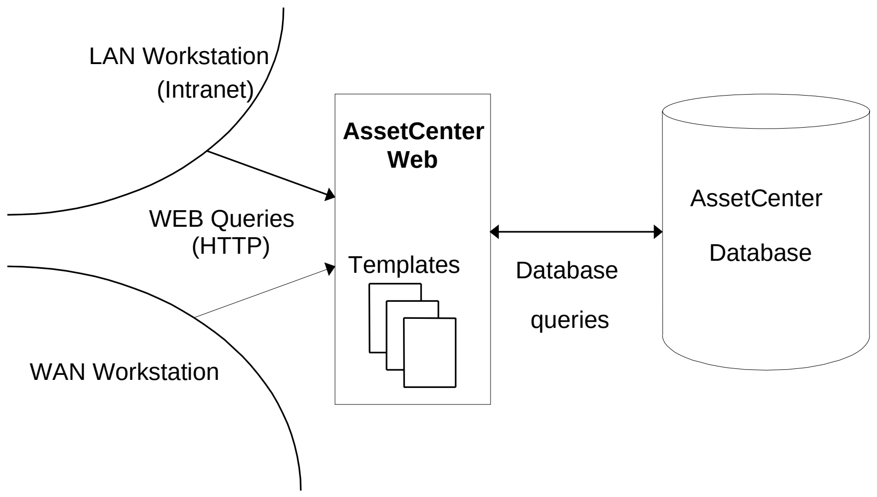
- AssetCenter Web provides an alternative interface to access the AssetCenter database. The methods used are identical to those in the standard AssetCenter user interface, in particular concerning integrity and security checks.

  Data manipulation follows the same security and integrity rules.

- Users are identified by their “Login”. Logins are identical to those used in the AssetCenter database (SQL name of the table: "amEmplDept", SQL name of the field: "UserLogin"). After logging in, users are associated with their dedicated resources so they can work in a private session, independently of other users. This mechanism provides access control to the AssetCenter database.
- To enable the creation of dynamic documents, AssetCenter Web uses a technique called "Server Side Scripting": all data processing operations are performed on the server. Only the resulting HTML pages are sent to the client. No processing is performed on the client workstation, except for HTML text rendering. This has several advantages: no special browser is required, information is compact and quickly displayed.

System structure

AssetCenter Web is provided as a Web server (HTTP) or an ISAPI module, and a set of documents called “templates” which define the pages accessible to users.



Overview of AssetCenter Web

Templates

Templates are basic documents on the AssetCenter Web server.

These documents comply with the HTML 3.2 standard.

They contain:

- Script sections: These are BASIC programs, which for example can collect data from the AssetCenter database.
- HTML text. It may be created by a Web page design tool, provided that the tool can preserve the script sections.

In the HTML sections, you can use a special notation to recover the value of variables. This notation has the following format: $$(variable name).

Template processing

When a user requests the system to display an AssetCenter Web page, the server accesses the corresponding template and processes it.

The template is analyzed sequentially: script sections are executed by a BASIC engine, and HTML sections are sent to the client after the system filters variables marked with the $$ characters.

Script sections are used to access the AssetCenter database and to perform operations on data. In general, script sections instantiate variables.

HTML sections define how the values of those variables will be presented on the user's screen, and how user pages will be presented in general. Before sending a line of HTML text to the client browser, AssetCenter Web replaces all the occurrences of $$(variable name) with the value of the variables.

User sessions

An AssetCenter Web user's first operation is to log on to the database by providing a "Login" and a valid password.

If the login is successful, AssetCenter Web assigns a session key to the user. This key is contained in all the links in the user's environment. It is used to control all the documents accessed.

Templates allow users to obtain the information which their AssetCenter profiles authorize them to access. Thus, users can view the list of purchase requests if they are allowed to view the records in the table of purchase requests.

NT service or ISAPI module

AssetCenter Web is available:

- As a Windows NT service. In this case, AssetCenter Web is a stand-alone server.

  This format is designed for enterprises needing a standard product requiring neither custom development nor set up. It is also suitable for enterprises with no other Web servers.

- As an ISAPI ("Internet Server Application Programming Interface") module: ISAPI technology makes it possible to integrate the AssetCenter Web server in Microsoft servers or ISAPI-compatible servers (Apache, etc.). In this case the AssetCenter Web server runs within an existing server.

  This form is particularly well suited for enterprises wishing to benefit from the advantages of ISAPI servers: Security, user authentication, real-time document encoding, additional tools, etc. This technology also allows users to run a single Web server.

Chapter 2 - Programming

Template structure

Templates comply with the HTML 3.2 standard. They consist of:

- BASIC script sections,
- pure HTML text.

Script sections

Script sections are delimited by the following tags:

```
<SCRIPT LANGUAGE="AmScript1.0">
…
</SCRIPT>
```

They are executed by the BASIC engine built into the AssetCenter Web server whenever a document is requested.

These sections contain all the code required to recover information in the database: Database queries, reading or writing records, etc. They also declare variables.

If the script writes to the standard output ("print"), data sent appears in the resulting HTML document. This makes it possible to dynamically build HTML from a script section.

Example of a simple script

The following example contains a simple script to logs in to the AssetCenter database:

```
<script language="Amscript1.0">
if AmwCurrentCnx() = 0 then
    iRc = AmwLogin(DbName, User, Password)
  if AmwCurrentCnx() <> 0 then
    print "Login OK"
  else
    print "Login failed"
  end if
end if
</script>
```

The program checks that no other logins are active ("AmwCurrentCnx() = 0"), then attempts to log in to the database.

The "DbName", "User", and "Password" variables must be instantiated with values representing a valid login profile. For example, these values may come from a login form where the user selects a database name, then enters a “Login” and password.

Note: The page following the identification request (entering the “Login” and password) must set up a valid connection to an AssetCenter database. Otherwise, the user session is rejected and the user is returned to the home page. When the login is completed, the user can continue the session and access the AssetCenter data.

Suggestions

You can create as many script sections as you like, and place them anywhere in the template.

It is preferable to use a single script section, however, and to put it at the start of the document, for performance and code legibility reasons.

If you are using an HTML editor, place the script section outside the body of the HTML document in order to avoid interpretation problems.

HTML

Templates are processed line by line, from top to bottom. Script sections are isolated and executed in a BASIC engine. HTML sections are sent line by line, after replacing variables by the values available at the time of processing.

Variables are identified by a special notation: $$(variable name). In general, values are set by the script section located at the start of the document.

Users receive the resulting HTML document. This document is compatible with the HTML 3.2 standard.

Sample template:

```
<script language="Amscript1.0">
  iRc = AmwFetchQuery("select Name from amEmplDept",0,2)
</script>

Employee:  $$(name0)
Employee:  $$(name1)
```

This example demonstrates the use of two variables containing the names of the first two employees in the employees and departments table. The rest of this manual describes the use of the script language in greater detail.

Variable-space

Principles

Each template includes a variable-space.

The variable-space is a storage area where you can store values identified by a name.

There are two types of variables with different scopes:

- Variables whose scope covers the entire session.
- Variables whose scope is limited to the template.

Session variables

These are global variables for the user session.

The information they contain is accessible throughout the session. They are visible to all templates.

Session variables may be used for example to store information concerning the user, security information, etc.

As with programming, it is best to limit the amount of global information. We recommend against defining too many session variables.

Template variables

These are local variables that are valid while processing the template only. These variables are instantiated in the script sections.

For example, a script section can execute queries on the AssetCenter database and obtain the results in the form of a list of indexed variables. This method is used to build tables presenting data extracted from an AssetCenter database table.

The following example extracts the first five employees from the employees and departments table, in alphabetical order:

```
<script language="Amscript1.0">
  iRc = AmwFetchQuery("select Name,FirstName from amEmplDept where
lEmplDeptId <> 0 order by Name",0,5)
</script>

<TABLE>

<TH> <TD> Name </TD> <TD> First name </TD> </TH>

<TR> <TD> $$(Name0) </TD> <TD> $$(FirstName0) </TD> </TR>
<TR> <TD> $$(Name1) </TD> <TD> $$(FirstName1) </TD> </TR>
<TR> <TD> $$(Name2) </TD> <TD> $$(FirstName2) </TD> </TR>
<TR> <TD> $$(Name3) </TD> <TD> $$(FirstName3) </TD> </TR>
<TR> <TD> $$(Name4) </TD> <TD> $$(FirstName4) </TD> </TR>
</TABLE>
```

The variables "Name0",...,"Name4" and "FirstName0",...,"FirstName4" are instantiated in the script section; they are local template variables.

Precedence rule

As with BASIC, local variables have precedence over global variables, and template variables have precedence over session variables.

Therefore the system first looks for a variable among the template variables; if not found, it looks in the global variables.

If a variable does not exist, the result is an empty string.

Note on variables in script areas

Script areas may also contain variables that behave in the same way as variables in a BASIC program:

- A local variable in a procedure is visible in the procedure only.
- A variable defined in a section is valid throughout the section.

By default, a variable defined in a script section is not visible outside that section. Functions are available to extend the scope of variables defined in script sections.

Defining the value of a variable

You can expose variables in the template or the user session using two functions:

- **`AmwSetTpl(<variable name>, <variable value>)`**: This function takes two parameters. The first is the name of the template variable whose variable you want to define. The second is the text value to assign to that variable.
- **`AmwSetEnv(<variable name>, <variable value>)`**: This function takes two parameters. The first is the name of the session variable whose variable you want to define. The second is the text value to assign to that variable.

Note: To make AssetCenter more easily accessible, most functions automatically instantiate template variables containing the result of operations. For example, the "AmwFetchQuery" function performs a query on the AssetCenter database and stores the query result in the template variables. Please refer to the manuel entitled "AssetCenter: Programmer's Reference" for more details on these functions.

Accessing variables

You can access a template or a session variable in two ways:

- for HTML text, simply call the variable using the following format: $$(variable name). The variable is replaced by its value when the template is processed.

  If the variable does not exist, it is replaced by an empty string.
- for script sections, two AssetCenter Web-specific functions are provided to obtain the text value of variables:
  - ❖ **`AmwGetEnv(variable name)`**: Returns the value of the session variable whose name is given as a parameter to the function.

- **`AmwGetTpl(variable name)`**: Returns the value of the template variable whose name is given as a parameter to the function.

Note: To avoid naming conflicts between script section variables and variables created automatically by certain APIs, template variables are not visible by default as BASIC variables. Therefore you must use the "AmwGetTpl" function to access the results of those functions.

Examples

The example below manually creates four variables in the local template space:

```
<script Language="Amscript1.0">

for i = 0 to 3
  varName = "Variable"&str(i)
  varValue = "Value"&str(i)
  AmwSetTpl varName,varValue
next i

</script>

<UL>
<LI> Variable0: $$(Variable0)
<LI> Variable1: $$(Variable1)
<LI> Variable2: $$(Variable2)
<LI> Variable3: $$(Variable3)
</UL>
```

The user sees the following document:

```
Variable0: Value0
Variable1: Value1
Variable2: Value2
Variable3: Value3
```

Each occurrence of the $$(variable name) notation has been replaced by the value defined in the script section.

In the following example, the script section accesses variables resulting from an API call, and creates new values from those variables to enhance the presentation of data on the user's screen.

```
<script language="Amscript1.0">
  iRc = AmwFetchQuery("select Name from amEmplDept where lEmplDeptId <>
0 order by Name",0,4)

  for i = 0 to 3
    varName = "Name"+str(i)
    varValue = AmwGetTpl(varName)
    newVarName = "Item"+str(i)
    AmwSetTpl newVarName,"<LI> "+varName+": "+varValue
  next i
</script>

<UL>
 $$(Item0)
 $$(Item1)
 $$(Item2)
 $$(Item3)
</UL>
```

In this case, the variables "Item0"..."Item3" contain both the information from the database and (<LI>) HTML tags.

To simplify templates and avoid redundancy in the HTML part of the document, you can create HTML in the script sections and place it in variables. This makes it easier to create tables and links. Ultimately, the HTML section could contain a single variable containing all the HTML code.

Note: We advise not using extremely large variables to avoid peaks in memory usage.

Here is what the user sees:

```
. Name0: Admin
. Name1: Los Angeles Agency
. Name2: Alex
. Name3: Helpdesk
```

Default System Variables

To make certain information permanently available, the AssetCenter Web server automatically creates a set of variables in the template space before the template is processed.

These variables provide information on the execution context. For example, they contain security keys, user identification, the name of the current document, etc.

They allow you to:

- display accurate information in case of errors.
- view the session key before creating links.

They may be used:

- in template script areas via the "AmwGetTpl" function.
- in HTML text. Simply call them in the following format: $$(variable name).

The following system variables are available:

- **`T_webport`**: port on the AssetCenter Web server.
- **`Sid`**: user session key.
- **`T_Query`**: current "query string".
- **`T_Referer`**: link in the source document used to access the current document.
- **`T_TemplateDoc`**: Name of the current document.
- **`T_TemplatePath`**: Full name of the current document (current document name and pathname).

Script language

The script language is a subset of BASIC:

- All BASIC operators and control structures are available.
- AssetCenter data is accessed via a programming interface (API). This interface is a list of functions directly accessible from BASIC.

The remainder of this section presents:

- The APIs.
- The script modules.

The APIs

To access the information contained in the AssetCenter database from template script sections, two functions are available:

- The first includes all the AssetCenter Web-specific functions.
- The second includes functions from the generic AssetCenter API.

AssetCenter Web–specific functions

These functions are prefixed by "Amw". They belong to one of two categories:

- data processing functions, independent of the database. Example: String concatenation functions.
- database management functions. Example: "AmwFetchQuery".

Some of these functions automatically instantiate variables in the template space. These variables are easy to use in HTML text or in code.

Example:

An AssetCenter Web-specific function runs a database query and recovers the results, e.g. a list of 10 assets identified by their asset tag.

The values of these items are defined as variables numbered automatically in the document space. The following variables are created: $$AssetTag0, $$AssetTag1, $$AssetTag2, ...$$AssetTag9. They may be used in HTML text.

For details on these functions, please refer to the manual entitled "AssetCenter: Programmer's Reference".

AssetCenter API functions

These functions are prefixed by "Am".

They allow you to work with the database: create or modify fields or links, execute queries, transactions, etc.

For more details on these functions, please refer to the manual entitled "AssetCenter: Programmer's Reference".

Script modules

A script module is a BASIC file with the ".bas" extension that contains functions you wish to use in several templates.

These functions are then imported in the script sections via the "import" command.

In the following example, the "Import errors.bas" command includes the "errors.bas" file in the script section. All the functions contained in this file are accessible in the section, including the "AmwErrorMsg()" function.

```
<script language="Amscript1.0">

  ' ==========================
  ' Utility functions
  ' ==========================
   import errors.bas

  ' ==========================
  ' Login
  ' ==========================
   If AmwCurrentCnx() = 0 Then
     iRc = AmwLogin(DbName, User, Password)

     ' Display errors
     If (iRc <> 0) Or (AmwCurrentCnx() = 0) Then print "<H1>";
AmwErrorMsg();"</H1>"
   End If
</script>
```

You can use these script modules as examples for your own programs. They are copied in in the "websrv\htdocs\tpl\" subfolder of the AssetCenter installation folder.

HTML forms

Definition

A form is a document used to hold information entered by a user on a client workstation: text entry box, selection box, check box, buttons to trigger operations, etc.

Forms are defined in the HTML 3.2 standard.

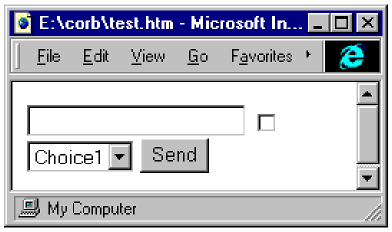
Forms are defined by:

- the list of data entry fields they contain (text entry fields, check boxes, etc.).
- an "action" field that designates a document on the server. This document receives all the information entered by the user if the user submits the document.

Sample form:

```
<form method =get action="gendata.htm" >
<input type=text name=user >
<input type=checkbox name=user > <br>
<select name=choice>
 <option value="Choice1"> Choice1 </option>
 <option value="Choice2"> Choice2</option>
</select>
<input type=submit name=go value="Send">
</form>
```

Here is how the user sees the form:



Window resulting from a form

AssetCenter Web form processing

AssetCenter Web uses forms as the basic mechanism for obtaining user information.

Examples: Entering a purchase order.

In general, forms are processed by calling a CGI ("Common Gateway Interface") program. When the form is submitted, this program receives the user information, decodes it, and generates the resulting document.

AssetCenter Web processes forms in a similar way:

- The "Action" field in each form designates a template.
- When processing the form, the template script area inherits all the variables that were defined in the form.

This method avoids:

- The need for an external application (CGI).
- Manual processing of information entered in forms.

You will find a sample form processing program for AssetCenter Web in the section entitled "Form processing" on page 27 in this manual.

Page composition

AssetCenter Web respects the HTML 3.2 standard. You can use any HTML editor to create your Web server pages.

These editors have the following advantages:

- They allow you to design pages graphically in WYSIWIG mode ("What You See Is What You Get").
- They also provide an HTML editor for working on areas that are not visible in preview mode (script areas, etc.).

Warning: Make sure you use an HTML editor that preserves script sections. The editor should not modify script sections.

Links

Links in AssetCenter Web

Links create connections between HTML documents.

With AssetCenter Web, users work in a private environment where document paths are prefixed by the session key.

For example, the URL ("Uniform Resource Locator") for the main menu is: "http://www.company.com/IDK-5454_52397424_8569/tpl/menu.htm".

The general structure for a document reference on the server is as follows:

```
<Session key>/<Document path>
```

The session key is used to control user access and guarantee security. It remains valid until the user logs out of the database, or until the defined timeout value is reached.

Everything runs as if the user was accessing a private document tree structure.

Creating links

All links in user pages must remain in the session context. There are two types of links:

- Relative links.
- Absolute links.

Relative links

Relative links specify a position in reference to the current document. This is the simplest case, because you do not need to specify the security key.

Examples:

"../ people.htm" points to a document called “people.htm” stored in the folder above the current document.

"home.htm" points to a document called "home.htm" stored in the same folder as the current document.

Absolute links

In this case, the link specifies the path starting from the root of the server. You must always create the link using the security key. This security key is stored in a SID system variable. In this case, you indicate the link as follows:

/$$(SID)/Full pathname for the document on the server.

When processing the template, the SID variable is replaced by the user session security key. The link displayed on the user screen is valid in the session context.

Warning: Absolute links that do not contain the session key are considered as an anonymous document request. This generally causes an error message, where AssetCenter Web indicates that it cannot send the document.

Suggestion

It is strongly recommended to use relative links only. This makes it easier to use standard HTML editors.

Debugging

To test the templates you have created, you must use a browser and request the documents via the AssetCenter Web server.

This allows you to view the results of the pages you have created and to locate any possible errors.

To determine the cause of a problem, you can have the system display the variables available in the template context, via the "AmwVars" function.

"AmwVars" displays the values of known variables in the document (both global and local variables) at the time when you call it. You can thus view the list of variables defined by an API function.

This function is useful for detecting syntax errors in $$(variable name) notations. This is especially useful because, by default, if a variable that does not exist is called using the $$(variable) format, it is replaced by an empty string.

Chapter 3 – Sample templates

Login

Here is a sample login script to an AssetCenter database:

```
<script language="Amscript1.0">

  ' ==========================
  ' Login
  ' ==========================
  If AmwCurrentCnx()=0 Then iRc = AmwLogin(DbName, User, Password)

    ' Display errors if any
    If (iRc <> 0) Or (AmwCurrentCnx() = 0) Then
      print "<H1> Login error:"; AmwErrorMsg();"</H1>"
    Else
      print "<H1> Login successful";AmwCurrentCnx();" </H1>"
    End If

  End If
</script>
```

The page displayed on the user screen may appear as follows:

```
Login successful cnx=20876816
```

This example:

- logs on to the database if no one else is logged on.
- displays a user message depending on whether or not a valid login identifier exists ("AmwCurrentCnx()") and depending on the return code from the login function. "AmwErrorMsg()" accesses the last error message in case of failure.
- automatically instantiates the AssetCenter Web "User" and "Password" variables using the data entered by the user in the login dialog box.

- The "DbName" variable containing the AssetCenter login name may come from a form.

  Here is a sample form used to select the AssetCenter login name:

```
<form method=get action="tpl/login.htm" >
<PRE>
 Database: <input type=text name=DbName > <input type=submit
value=Login name=Action>
</PRE>
</form>
```

  In this form, in order for AssetCenter Web to accept to process the target template ("login.htm") as a login template (able to set up a user session), you must provide an “Action” parameter containing the “Login” value. Use the submit button to provide these values.

  To use another submission method (a different button name, an icon to be clicked, etc.), you can place the “Action” parameter containing the "Login" value in a hidden field.

  In this case, the form for selecting the AssetCenter login name may be as follows:

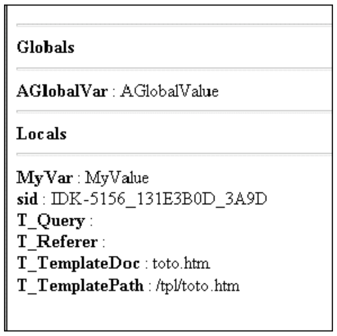
```
<form method=get action="tpl/login.htm" >
<PRE>
 Database: <input type=text name=DbName > <input type=submit
value=Go name=Button>
<input type=hidden value=Login name=Action>

</PRE>
</form>
```

Displaying variables

The example below shows how to create a template variable and a session variable:

```
<script language=”AmScript1.0”>
  AmwSetEnv “AGlobalVar”,”AGlobalValue”
  AmwSetTpl “MyVar”,”MyValue”
</script>
$$(AmwVars())
```

The $$(AmwVars()) command displays the list of global and local variables in the resulting document:



Globals

AGlobalVar : AGlobalValue

Locals

MyVar : MyValue
sid : IDK-5156_131E3B0D_3A9D
T_Query :
T_Referer :
T_TemplateDoc : toto.htm
T_TemplatePath : /tpl/toto.htm

Resulting page

Simple reading of records

The following example shows how to access a data set using an AQL query (AQL is the AssetCenter Query Language). For further information on AQL, please refer to "Reference guide: Administration and advanced use" chapter "Writing queries in AQL":

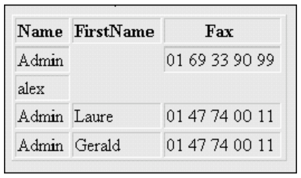
```
<script language="Amscript1.0">
 if AmwCurrentCnx() = 0 then iRc = AmwLogin(DbName, User, Password)
 iRc=AmwFetchQuery("select name,firstname,fax from amempldept where
lEmplDeptid>0 and bdepartment=0 order by name",0,4)
</script>

<TABLE Border=1>
<TR> <TH>  Name   <TH>    FirstName   <TH>   Fax      <TH> </TR>
<TR> <TD> $$(Name0) <TD> $$(FirstName0) <TD> $$(Fax0) <TD> </TR>
<TR> <TD> $$(Name1) <TD> $$(FirstName1) <TD> $$(Fax1) <TD> </TR>
<TR> <TD> $$(Name2) <TD> $$(FirstName2) <TD> $$(Fax2) <TD> </TR>
<TR> <TD> $$(Name3) <TD> $$(FirstName3) <TD> $$(Fax3) <TD> </TR>
</TABLE>
```

This example requests the first four records in the Departments and Employees table. The result is stored in the "Name0" … "Name4", "FirstName0" … "FirstName4", "Fax0" … "Fax4" variables.

The variable names correspond to the field names used in the AQL query. Please refer to the manual entitled "AssetCenter: Programmer's Reference" for further details on the "amwFetchQuery" function.

The variable names can be complex if the AQL query returns links or calculated fields. In this case, use the "AmwVars()" function after executing the query to obtain the field names.

Here is the page resulting from this template:



Name	FirstName	Fax
Admin		01 69 33 90 99
alex		
Admin	Laure	01 47 74 00 11
Admin	Gerald	01 47 74 00 11

Resulting page

Navigation

The example below adds another feature to the previous template: navigation. The page contains two links where the user can click: "Prev" and "Next" to display the previous or next results page.

This is performed using a template variable: “Pos”. This variable defines the position of the first record recovered by "AmwFetchQuery()".

```
<script language="AmScript1.0">

  iPos = val( AmwGetTpl("Pos") )
  iRc = AmwFetchQuery("select name,FirstName from amEmplDept where
lEmplDeptid>0 and bdepartment=0 order by name",iPos,4)
  iNextPos = iPos+4
  iPrevPos = iPos-4
  AmwSetTpl "NextPos",iNextPos
  AmwSetTpl "PrevPos",iPrevPos

</script>


<TABLE Border=1>
<TR> <TH>  Name   <TH>    FirstName   <TH>   Fax      <TH> </TR>
<TR> <TD> $$(Name0) <TD> $$(FirstName0) <TD> $$(Fax0) <TD> </TR>
<TR> <TD> $$(Name1) <TD> $$(FirstName1) <TD> $$(Fax1) <TD> </TR>
<TR> <TD> $$(Name0) <TD> $$(FirstName2) <TD> $$(Fax2) <TD> </TR>
<TR> <TD> $$(Name0) <TD> $$(FirstName3) <TD> $$(Fax3) <TD> </TR>
</TABLE>


<A href="navig.htm?pos=$$(PrevPos)"> Prev </A><br>
<A href="navig.htm?pos=$$(NextPos)"> Next </A>
```

During the first call, the "Pos" variable does not exist; the position is therefore null.

The HTML section creates two links, "Prev" and "Next", containing a parameter ("?pos=") defined by the current position. This gives access to the next or previous four records.

Form processing

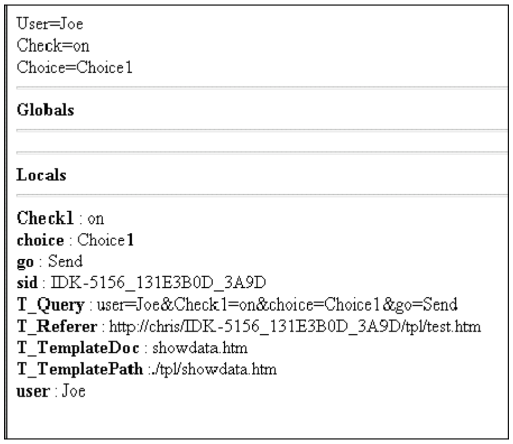
Here is a sample form:

```
<form method =get action="showdata.htm" >
<input type=text name=user >
<input type=checkbox name=check1 > <br>
<select name=choice>
<option value="Choice1"> Choice1 </option>
 <option value="Choice2"> Choice2 </option>
</select>
<input type=submit name=go value="Send">
</form>
```

The form causes the system to process the "showdata.htm" template shown below:

```
<script language="AmScript1.0">

  print "User=";user;"<br>"
  print "Check=";Check1;"<br>"
  print "Choice=";Choice;"<br>"

</script>

$$(AmwVars())
```

The resulting user page:


User=Joe
Check=on
Choice=Choice1

Globals

Locals

Check1 : on
choice : Choice1
go : Send
sid : IDK-5156_131E3B0D_3A9D
T_Query : user=Joe&Check1=on&choice=Choice1&go=Send
T_Referer : http://chris/IDK-5156_131E3B0D_3A9D/tpl/test.htm
T_TemplateDoc : showdata.htm
T_TemplatePath :./tpl/showdata.htm
user : Joe

Resulting page

Data entry forms with choices

When using forms, you sometimes need to indicate several document in order to be able to process a request according to an item the user selects with the mouse.

The HTML standard only defines the use of a single document. Its name is defined in the “action” field of the “FORM” tag.

AssetCenter Web allows you to associate an item in a form with a document other than the “action” document. To do that, you must give a special name to the item, using the following syntax:

```
Sub#<doc-name>#<indication>
```

The server attempts to load the document called "<doc-name>.htm" and assigns the value "indication" to the "T_Hint" variable.

This method is very practical when the values from a template need to be used in several documents

For an example of this technique, refer to the purchase request creation template "newrq_f.htm" (located in in the "websrv\htdocs\tpl\" subfolder of the AssetCenter installation folder). The "Requester", "Budget", "Cost center" and "Project" fields are associated with the buttons labeled "Sub#user_l#Select:lRequesterId,Requester", "Sub#budget_l#Select:lBudgId,Budget", "Sub#costc_l#Select:lCostId,CostCenter" and "Sub#proj_l#Select:lProjId,Project". They allow you to access the "user_l.htm", "budget_l.htm", "costc_l.htm" and "proj_l.htm" documents and select an item.

The programmer is responsible for keeping the user information already entered in the form by systematically sending the "Query String" string.

Creating records

The following sample script creates a record in the table of purchase requests.

```
iRc = AmwNewRecord("amRequest",
"ReqPurpose(PurposeId);Requester(RequesterId);Comment(CommentId)")
```

Updating records

The sample script below modifies the location of a record in the Assets table.

```
' try to update location
  iRc = AmwUpdateRecord("amAsset","lLocaId(LocaId)",AssetId)
```

Error handling

If an error occurs, you can access a detailed message describing the reasons why the operation failed. Use the Web API function "AmwErrorMsg()" to access the message text.

This message corresponds to the last operation executed, and is erased whenever another API call is made.

Chapter 4 – Advanced configuration

Manual server startup

The server is provided as an executable file called "amw3.exe".

This executable can be run as a service under Windows NT, or can be launched manually as a standard executable. Launching the program manually allows you to recover server error messages in a console window.

In service mode, the server is started and stopped via the Windows NT Control Panel.

For further information, please refer to the manual entitled "AssetCenter Web User's Guide", chapter "Implementing AssetCenter Web", section "Launching AssetCenter Web".

Server configuration

All the server configuration data is stored in the "amw3.ini" file.

For further information, please refer to "AssetCenter Web User's Guide", chapter "Administrating AssetCenter Web", section "Configuring AssetCenter Web via "amw3.ini".

Generic login

When operating normally, AssetCenter Web uses the HTTP authentication protocol to obtain the user ID and password. This method displays the login window where users enter their “Login” and password.

If you want to provide access to the AssetCenter Web server without requiring the user to enter the login information, you can disable this mechanism.

To do this, insert the "AllowNoAuth = 1 " option in the [GLOBAL] section of the "amw3.ini" file. This file is described in detail in "AssetCenter Web User's Guide", chapter "Administrating AssetCenter Web", section "Configuring AssetCenter Web via "amw3.ini".

In this case, users can access the first template without providing login information. However, the programmer is responsible for setting up a valid database login based on external information.

For example, you can use a generic "Login" name:

```
<script language="Amscript1.0">

  ' ==========================
  ' Generic login using Guest account
  ' ==========================
  If AmwCurrentCnx()=0 Then iRc = AmwLogin(“MyDb”,”Guest”,””)

    ' Display errors if any
    If (iRc <> 0) Or (AmwCurrentCnx() = 0) Then
      print "<H1> Login error:"; AmwErrorMsg();"</H1>"
    Else
      print "<H1> Login successful";AmwCurrentCnx();" </H1>"
    End If

  End If
</script>
```

In the example above, a guest account with no password allows users to connect to the "MyDb" database.

Template Batch Processing

In some cases, template processing time may not be compatible with real-time usage.

This is true for documents that perform numerous database accesses to create complex pages.

In this case, you can prepare these documents off-line from corresponding templates.

Launch the server manually and specify the document to process along with additional information, using the following syntax:

```
amw3 -template:<template file> [-qstr:<query string>]
```

- the "template" argument defines the name of the template to process.
- the optional "qstr" argument defines the additional parameters as a query string; use the same format as when submitting forms via CGI.

Example:

```
Amw3 –template:huge.htm qstr:”DbName=MyDb&Usr=Guest&Pwd=” > res.htm
```

In this example, the "huge.htm" template is processed and the resulting document is directed to the "res.htm" file.

The additional query string provides the following information:

- DbName = "MyDb"
- Usr = "Guest"
- Pwd = ""

This information is provided through variables in the template data space.