Development Exchange Summit 2009

CA Workload Automation DE– JavaScripting

David A. Leigh – Principal Consultant - Automation

SHIFT YOUR AUTOMATION INTO HIGH GEAR

Terms of This Presentation

For Informational Purposes Only

Agenda

> JavaScript – What is it & Why does DE use it.
> Product Specific JavaScript Functions
> Variables Review
> Troubleshooting & Logging
> Q & A

JavaScript – What is it & Why does DE use it?

> JavaScript is a scripting language commonly used to create dynamic HTML pages that process user input.
> With CA Workload Automation DE, JavaScript supports many functions that are unavailable from the CA Workload Automation DE Desktop Client interface.
> By using JavaScript scripts within your Applications, you can take advantage of the server's advanced scheduling features.
> You can use JavaScript scripts to:
  - Create and manipulate symbolic variables.
  - Use CA WA built-in functions.
  - Perform comparison, arithmetic, and logical operations.
  - Prepare program input and parameters.
  - Build decisions into schedules.
> The CA Workload Automation DE Server uses JavaScript release 1.5 as its internal scripting language and conforms to Edition 3 of the ECMA-262 Standard for scripting languages.

JavaScript – What is it & Why does DE use it?

- The CA Workload Automation DE product has included a full version of JavaScript 1.5 for the products use.
- There are no changes to how the JavaScript implementation operates within the product as it would outside of the product.
- There are specific features and functions that have been added to the JavaScript implementation to facilitate interfacing to the product
- The use of JavaScript was expected to be used as an IF/THEN/ELSE engine to facilitate advanced features of the product.
- The implementation is not expected to be used for all of the functions that JavaScript can perform.

JavaScript – What is it & Why does DE use it?

> Suggested Reading Materials & Websites:

- CA Workload Automation DE Programming Guide
- www.javascript.com – The self proclaimed "Definitive JavaScript Resource" – General JavaScript Examples & JavaScript programming forum.
- JavaScript: The Definitive Guide - by David Flanagan
- JavaScript: A Beginner's Guide, Second Edition - by John Pollock

Choosing where to specify the script

> You can specify the script within an Event definition, an Application definition (Application level), or a job definition (job level).

> Where you specify the script determines the availability of elements such as variables, symbolic variables, and parameters.

Choosing to specify scripts within an Event definition

> Specify the scripts within an Event definition for the following scenarios:

- You want to run multiple scripts when the Event is triggered. In contrast, the Application definition and job definition let you specify only one script using the **At run time** option and one script using the **At Event trigger time** option.

  Note: Specifying one script in the Event definition is equivalent to specifying the script in the Application definition using the **At Event trigger** time option.
- The script defines or sets values for system-level symbolic variables. The names of these variables begin with the prefix ESP.
- The script defines or sets values for Application-level symbolic variables. The names of these variables begin with the prefix APPL.

Choosing to specify a script within an Application definition

> Specify the script within an Application definition for the following scenarios:
  - The script defines or sets values for symbolic variables that need to be available to multiple jobs within the Application. These variables can be system-level (the names begin with the prefix ESP) or Application level (the names begin with the prefix APPL).
  - The script passes parameters from the Event to the Application.
  - The script determines whether multiple jobs will run.
  - The script generates date and time variables to be used by multiple jobs.
  - The script defines or sets values for job-level symbolic variables. The names of these variables begin with the prefix WOB. For these scripts, you must select the At run time option. The script runs when each job in the Application starts to run.

Choosing to specify a script within a job definition

> Specify the script within a job definition for the following scenarios:

- The script defines or sets values for symbolic variables that begin with the prefix WOB.
- The script sets values for variables used by a single job.
- The script sets a variable whose value must be confined to a single job (the symbolic variable is used in multiple jobs but must have a unique value for each job).
- The script specifies run criteria for a single job.
- The script defines or sets values for system-level symbolic variables. The names of these variables begin with the prefix ESP.
- The script defines or sets values for Application-level symbolic variables. The names of these variables begin with the prefix APPL.

Choosing when to run the script

> You can choose to run a script at Event trigger time or when one or more jobs run (at run time). When you run the script depends on the kinds of values you are calculating within the script. Some information is required at trigger time (when the Application builds) and other information is required at run time.

> For example, when an Event triggers and the CA WA server builds the Application, the server needs to know which jobs will be run as part of that Application. This means that all of the run frequencies need to be resolved at Event trigger time. The server, however, does not need to know the argument being passed to a UNIX script until the script is ready to run. This means that arguments do not need to be resolved until run time.

> **Note:** If you plan to use a symbolic variable as part of a job name or qualifier, the symbolic variable must be assigned a value at Event trigger time.

Properties Resolved at Trigger time

Properties	CA WA Desktop Client dialog	CA WA Desktop Client fields
> Application name	> Application properties dialog	> Name dialog
> Default Agent name	> Application properties dialog	> Agent dialog
> Application run frequency	> Application properties dialog	> Run frequency section
> Job name and qualifier	> Job definition Basic dialog	> Name and Qualifier fields
> Job run frequency	> Job definition Basic dialog	> Run frequency section
> Notifications	> Job definition Notifications dialog	> All fields within the Alerts tab
> Job resources	> Job definition Resources dialog	> All fields
> Job time dependencies	> Job definition Time Dependencies dialog	> All fields
> External job attributes	> External job definition Basic dialog	> All fields

Properties Resolved at Run time

Properties	CA WA Desktop Client dialog	CA WA Desktop Client fields
> Email addresses	> New Email Notification dialog	> To field
> Agent specifications	> Job definition Basic dialog	> Agent name, Command to run, Script/command name, Arguments to pass, and User ID fields
> Environment variables	> Job definition Environment Variables dialog	> Name and Value fields
> OS/400 environment specifications	> Environment dialog	> Library specifications, Job specifications, and OS/400 exit program fields

Properties Resolved at Run time

Properties	CA WA Desktop Client dialog	CA WA Desktop Client fields
> Exit codes	> Job definition Exit Codes tab	> Code and Interpretation fields
> Trigger conditions for File trigger jobs	> Job definition Basic dialog > Job definition User/Group specifications dialog	> File name field > Owner user ID, group, and Monitor as user fields
> Step specifications for SAP R/3 jobs	> Job definition Step Specifications dialog	> All fields
> Agent specifications for PeopleSoft jobs	> Job definition Basic dialog	> Process name and Process type fields

Storing the script

> Before the script can be available to an Event, Application, or job, you must first store the script in the Application that will use it or in the JavaScript repository on the CA WA server.

> When storing the script, you can write the script in CA WA Desktop Client or import the script from your local computer or a network drive. The name of each script must be unique for every instance of the server.

Storing the script

> If a JavaScript script is used in only one Application, you can store the script in the Application by using CA WA Desktop Client.

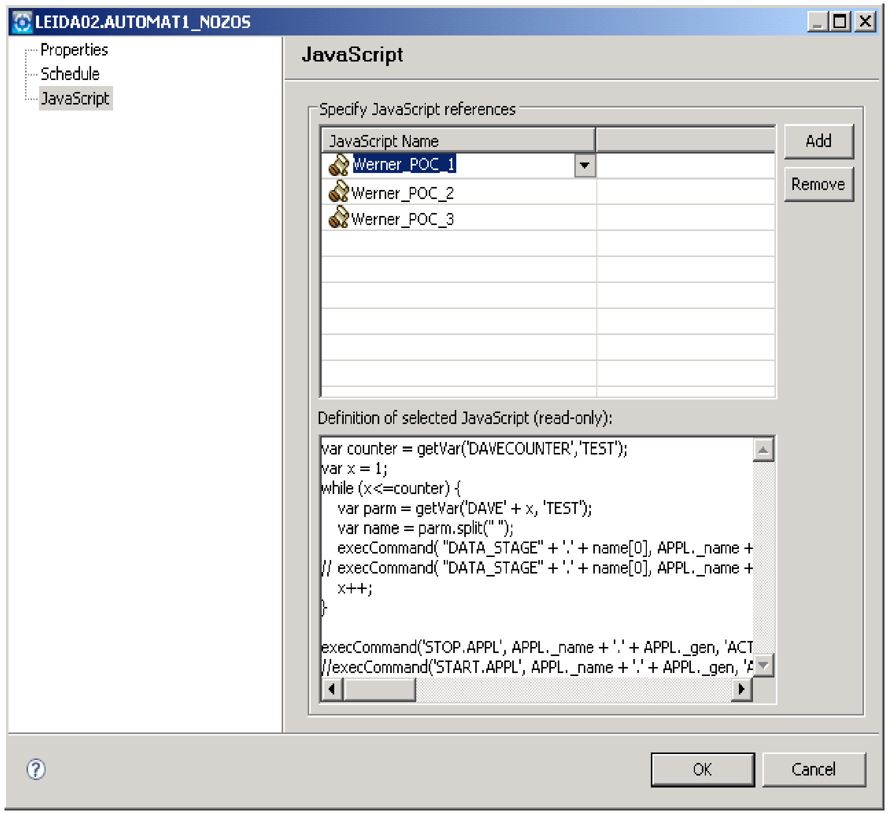
> If you want to use the script in multiple Applications, store the script in the JavaScript repository on the CA WA server.

> The JavaScript repository provides a common storage location on the CA WA server for your JavaScript scripts.

Specifying scripts within an Event definition

> You can specify a list of JavaScript scripts in an Event definition, which are run when the Event is triggered.

> You can specify scripts in an Event definition to set default values for the Application, define symbolic variables to be used by the Application, and for many other purposes.



JavaScript Name	
Werner_POC_1	
Werner_POC_2	
Werner_POC_3	

Definition of selected JavaScript (read-only):

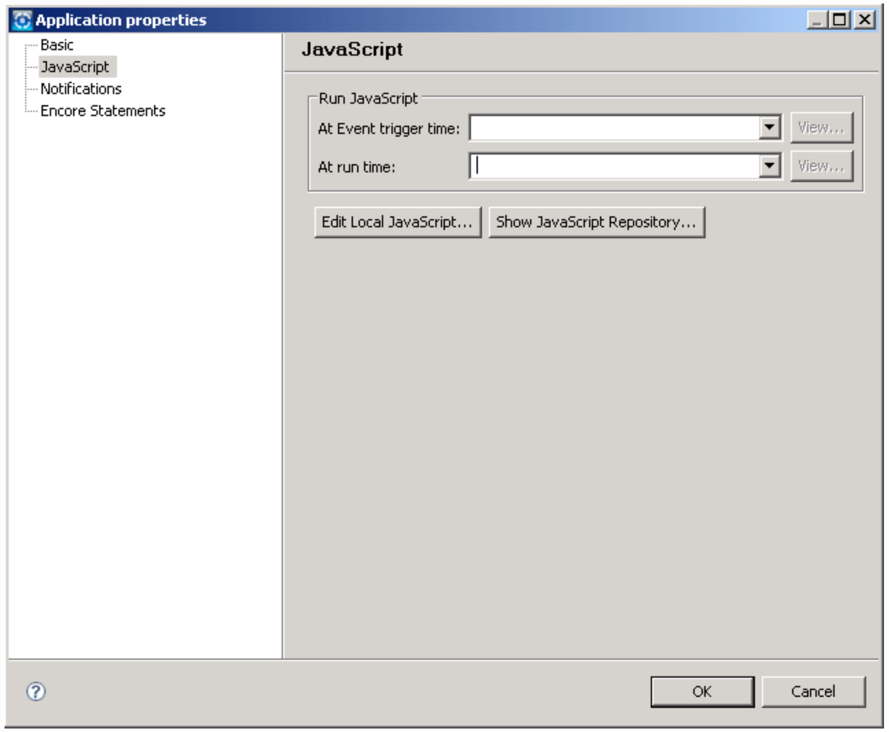
```
var counter = getVar('DAVECOUNTER','TEST');
var x = 1;
while (x<=counter) {
   var parm = getVar('DAVE' + x, 'TEST');
   var name = parm.split(" ");
   execCommand( "DATA_STAGE" + '.' + name[0], APPL._name +
// execCommand( "DATA_STAGE" + '.' + name[0], APPL._name +
   x++;
}

execCommand('STOP.APPL', APPL._name + '.' + APPL._gen, 'ACT
//execCommand('START.APPL', APPL._name + '.' + APPL._gen, 'A
```

Specifying a script within an Application definition

> You can specify a JavaScript script within an Application definition, and run it when the Event is triggered or when each job in the Application starts to run.

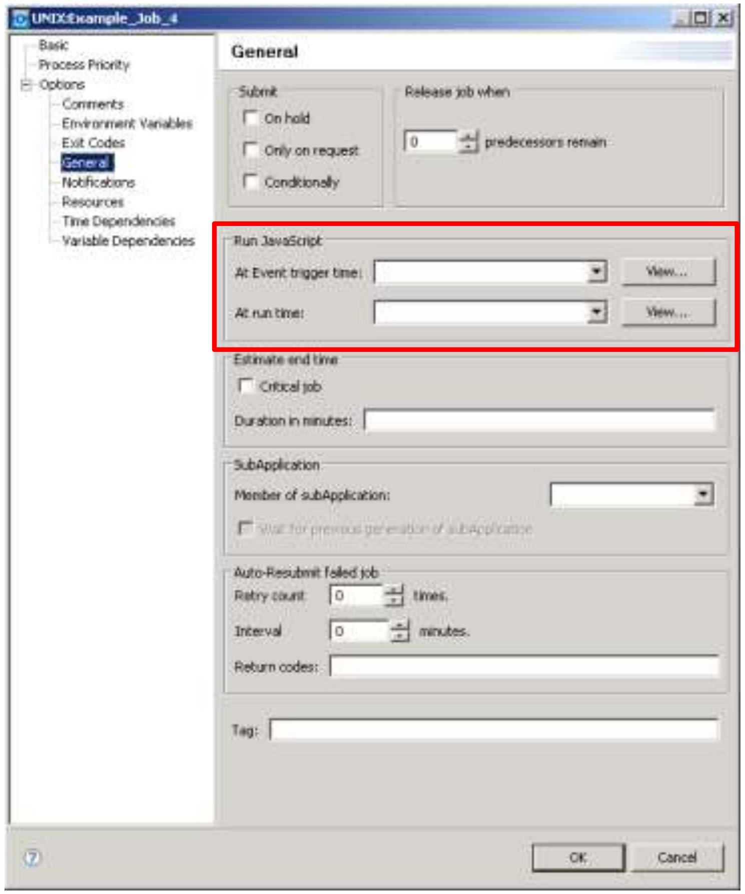
> You can specify a script in an Application definition to set default values for the Application, define symbolic variables to be used by the Application, and for many other purposes.



Specifying a script within a job definition

> You can specify a JavaScript script within a job definition, and run it when the Event is triggered or when the job runs.

> You can specify a script in a job definition to create symbolic variables for use by the job, specify alternative times to submit the job, and for other purposes.



Submit
- [ ] On hold
- [ ] Only on request
- [ ] Conditionally

Release job when: 0 predecessors remain

Run JavaScript
At Event trigger time:
At run time:

Estimate end time
- [ ] Critical job

Duration in minutes:

SubApplication
Member of subApplication:
- [ ] Wait for previous generation of subApplication

Auto-Resubmit failed job
Retry count: 0 times.
Interval: 0 minutes.
Return codes:

Tag:

Product Based JavaScript Functions

> A function is a set of instructions that can receive data, process that data, and return a value.

> The CA WA server provides built-in functions you can use within a JavaScript script.

> Built-in functions do not have a symbol introducer character—they are JavaScript constructs that are recognized by the server automatically.

> A built-in function appears in the following format function_name(arguments) where function_name must be a valid function name and arguments is a list of values specific to that function.

> Function names are case sensitive and must be typed exactly as shown. Unless otherwise specified, parameters and keywords provided in the syntax are also case sensitive.

Product Based JavaScript Functions

> Using built-in functions

- You can use CA WA server built-in functions in the following ways:
  - Within a script that is run at the Application level to have the returned values available throughout the Application.
  - Within a script that is run at the job level. In this usage, the returned values can be restricted only to that job.
  - Within a script as part of an Alert definition.
- You can use built-in functions with conditional logic to cause different actions to occur based on circumstances.

Product Based JavaScript Functions

> Using built-in functions

- For example, if today is Friday, you want joba to complete successfully before 5:00 pm. On the remaining days, joba must complete successfully by 9:00 pm. You can use the built-in function today in a job-level script as follows:

```
if (today('friday')) WOB.dueout = '5pm‘;
else WOB.dueout = '9pm';
```

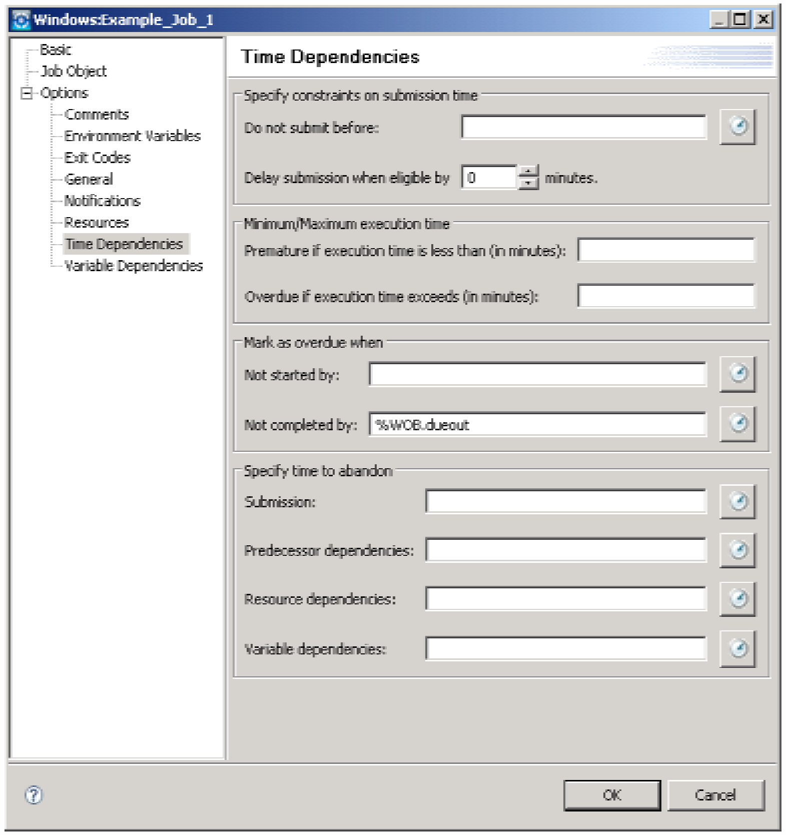
- Another way to specify the same condition is by using the JavaScript condition (?) as follows:

```
WOB.dueout = today('friday') ? '5pm' : '9pm';
```

In this example, the server assigns the WOB.dueout variable a value depending on the day of the week.

Product Based JavaScript Functions

> Using built-in functions
  - To complete the example, you specify %WOB.dueout on the job's Time Dependencies dialog in the Not completed by field.
  - On Fridays, %WOB.dueout resolves to 5pm, which is the time the job must complete successfully before it is marked overdue.
  - On all other days, %WOB.dueout resolves to 9pm, which is the time the job must complete successfully before it is marked overdue.



Time Dependencies

Specify constraints on submission time
Do not submit before:
Delay submission when eligible by 0 minutes.

Minimum/Maximum execution time
Premature if execution time is less than (in minutes):
Overdue if execution time exceeds (in minutes):

Mark as overdue when
Not started by:
Not completed by: %WOB.dueout

Specify time to abandon
Submission:
Predecessor dependencies:
Resource dependencies:
Variable dependencies:

Product Based JavaScript Functions

> Categories of built-in functions

- Built-in functions fall into the following categories.
  - **Calendaring functions** - Perform actions based on schedule criteria, calculate time periods, and make conditional logic statements to control the occurrence of actions.
  - **Event trigger functions** - Control the triggering of Events.
  - **File-level functions** - Perform file-level operations against individual files and return the results.

Product Based JavaScript Functions

> Categories of built-in functions

- Built-in functions fall into the following categories.
  - **Integer functions** - Format any number to an integer.
  - **Job, subApplication, and Application control functions** - Control a specified job, subApplication, or Application.
  - **Resource functions** - Reset resource availability.
  - **Symbolic variable functions** - Determine which symbolic variables exist and create user-defined date and time symbolic variables.

Product Based JavaScript Functions

> Calendaring functions

- These functions perform actions based on schedule criteria, calculate time periods, and make conditional logic statements to control the occurrence of actions.

Calendaring Functions

Function Name	Function Description
> daysBetween	> Use the daysBetween function to calculate the amount of time between two dates. The time can be calculated in days, months, workdays, and so on. Specify a start and end period and the type of date to use in the calculation.
> daysFrom	> Use the daysFrom function to calculate the number of days from a date you specify to today. Specify a schedule expression that resolves to a date.
> daysTo	> Use the daysTo function to determine the number of days from today to a date you specify. Specify a schedule expression that resolves to a date.
> today	> Use the today function with conditional logic to cause an action to occur. The today function compares a schedule expression you specify to today's date. The CA Workload Automation DE server returns a value of true or false, depending on whether the expression resolves to today's date.
> tomorrow	> Use the tomorrow function with conditional logic to cause an action to occur. The tomorrow function compares a schedule expression you specify to tomorrow's date. The CA Workload Automation DE server returns a value of true or false, depending on whether the expression resolves to tomorrow's date.

Calendaring Functions

Function Name	Function Description
> yesterday	> Use the yesterday function with conditional logic to cause an action to occur. The yesterday function compares a schedule expression you specify to yesterday's date. The CA Workload Automation DE server returns a value of true or false, depending on whether the expression resolves to yesterday's date

Event trigger function

Function Name	Function Description
> execTrigger	> Use the execTrigger built-in function to automatically trigger an Event from within a JavaScript script.
	Function Syntax
	execTrigger('eventname','ADD | REPLACE','time','NOHOLD | HOLD`, 'userparameter1','userparameter2','userparameter3`,'userparameter4','rootjob');
	Example Code
	> Triggering an Event execTrigger('CYBER.BACKUPS');
	> Triggering an Event at a specific time execTrigger('CYBER.PAYROLL','replace','4pm');
	> Triggering an Event using the hold parameter execTrigger('CYBER.RECOVER','','','hold');

File-level Functions

Function Name	Function Description
> file_appendContents	> Adds new data to the specified file. The function returns a Boolean value. True is returned if the contents were successfully added to the specified file. False is returned if an IOException or FileNotFoundException exception occurs.
> file_canRead	> Tests whether the application can read the specified file. The function returns a Boolean value. True is returned if the specified file exists and it can be read by the application. False is returned otherwise.
> file_canWrite	> Tests whether the application can modify the specified file. The function returns a Boolean value. True is returned if the file system contains the specified file and the application is allowed to write to the file. False is returned otherwise.
> file_create	> Automatically creates a new, empty file if the file does not exist. The function returns a Boolean value. True is returned if the file does not exist and the file is created. False is returned if the specified file already exists.
> file_delete	> Deletes the specified file or directory. A directory must be empty to be deleted. The function returns a Boolean value. True is returned if the file or directory is successfully deleted. False is returned otherwise.

File-level Functions

Function Name	Function Description
> file_exist	> Tests whether the specified file or directory exists. The function returns a Boolean value. True is returned if the file or directory exists. False is returned otherwise.
> file_getLastLine	> Gets the last line of the specified file even if the line is empty. The function returns a string. If successful, this function returns the content of the last line or, if unsuccessful, [undefined] is returned.
> file_getLastModified	> Returns the time that the specified file was last modified. The function returns a number. If successful, this function returns the time that the file was last modified, measured in milliseconds since the epoch (00:00:00 GMT, January 1, 1970). The value 0L is returned if the file does not exist or if an I/O error occurs. If the task was unsuccessful, then [undefined] is returned.
> file_getLastNonEmptyLine	> Gets the last non-empty line of the specified file. The function returns a string. If successful, this function returns the content of the last line or, if unsuccessful, [undefined] is returned.

File-level Functions

Function Name	Function Description
> file_getLength	> Returns the length of the specified file. The function returns a number. If successful, the function returns the length, in bytes, of the file specified by the file path. The value 0L is returned if the file does not exist. The return value is unspecified if the file path denotes a directory. If the function was unsuccessful, [undefined] is returned.
> file_getLineCount	> Gets the number of lines in a specified file. The function returns a number. If successful, this function returns the number of lines or, if unsuccessful, [undefined] is returned.
> file_getName	> Returns the name of the specified file or directory. This is just the last name in the pathname's sequence. If the pathname's sequence is empty, the empty string is returned. The function returns a string. If the function is unsuccessful, [undefined] is returned.
> file_getParent	> The parent of an abstract pathname consists of the pathname's prefix, if any, and each name in the pathname's sequence except for the last. If the pathname's sequence is empty, the pathname does not name a parent directory.

File-level Functions

Function Name	Function Description
> file_isDirectory	> Tests whether the specified file is a directory. The function returns a Boolean value. The function returns true if the specified file exists and it is a directory. False is returned otherwise.
> file_isFile	> Tests whether the specified file is a normal file. A file is normal if it is not a directory. The function returns a Boolean value. The function returns true if the specified file exists and it is a normal file. False is returned otherwise.
> file_isHidden	> Tests whether the file named by this abstract pathname is a hidden file. The definition of hidden depends on the operating system. On UNIX systems, a file is hidden if the file name begins with a period character ('.'). On Microsoft Windows systems, a file is hidden if it has been marked as such in the file system. The function returns a Boolean value. The function returns true if the specified file is hidden. False is returned otherwise.
> file_list	> Returns an array of strings naming the files and directories in the directory specified by the file path. If this path does not denote a directory, this function returns [undefined]. Otherwise an array of strings is returned, one entry for each file or directory in the file path. The array will be empty if the specified directory is empty.

File-level Functions

Function Name	Function Description
> file_mkdirs	> Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories. Even if this operation fails, it may have succeeded in creating some of the necessary parent directories. The function returns a Boolean value. The function returns true if the directory is created, including any necessary parent directories. False is returned otherwise.
> file_read	> Reads the content of a file, byte by byte. A start offset can be specified to skip an offset number of bytes. A length can be specified to limit the data that can be read. The function returns a string. If successful, this function returns the content of the file or, if unsuccessful, [undefined] is returned.
> file_readLines	> Reads the content of a specified file line by line. A start offset can be specified to skip an offset number of lines. A line count can be specified to limit the data that can be read. If successful, this function returns an array containing the lines that were read or, if unsuccessful, [undefined] is returned.
> file_renameTo	> Renames the specified file. Whether this method can move a file from one file system to another is platform-dependent. The return value should always be checked to make sure that the rename operation was successful. The function returns a Boolean value. True is returned if the renaming succeeded. False is returned otherwise.

File-level Functions

Function Name	Function Description
> file_replaceContents	> Overwrite the file with new data. The function returns a Boolean value. True is returned if the new data was written correctly. False is returned otherwise.

File-level Functions

Function Name	Example Code
> file_appendContents	> APPL.append = file_appendContents('/home/reports/logfile1', 'new data');
> file_canRead	> APPL.status = file_canRead('c:\\batch\\myfile.txt');
> file_canWrite	> APPL.status = file_canWrite('/home/reports/logfile1');
> file_create	> APPL.newfile = file_create('c:\\batch\\myfile.txt');
> file_delete	> APPL.status = file_delete('/home/reports/logfile1');
> file_exist	> APPL.status = file_exist('c:\\batch\\myfile.txt');
> file_getLastLine	> APPL.lastLine = file_getLastLine('/home/reports/logfile1');
> file_getLastModified	> APPL.tlm = file_getLastModified('c:\\batch\\myfile.txt');
> file_getLastNonEmptyLine	> APPL.lastLine = file_getLastNonEmptyLine('/home/reports/logfile1');
> file_getLength	> APPL.length = file_getLength('c:\\batch\\myfile.txt');
> file_getLineCount	> APPL.lineCount = file_getLineCount('/home/reports/logfile1');

File-level Functions

Function Name	Example Code

> file_getName	> // returns 'myfile.txt‵
	APPL.fileName = file_getName('c:\\batch\\myfile.txt');

> file_getParent	> // returns '/home/reports‵
	APPL.parent = file_getParent('/home/reports/logfile1');

> file_isDirectory	> // returns false
	APPL.directory = file_isDirectory('c:\\batch\\myfile.txt');
	> // returns true
	APPL.directory = file_isDirectory('/home/reports/');

> file_isFile	> APPL.file = file_isFile('c:\\batch\\myfile.txt'); // returns true
	> APPL.file = file_isFile('/home/reports/'); // returns false

> file_isHidden	> // returns false
	APPL.hidden = file_isHidden('/home/reports/logfile1');
	> // returns true
	APPL.hidden = file_isHidden('/home/reports/.logfile');

File-level Functions

Function Name	Example Code

> file_list	> // returns ['logfile1', '.logfile']
	APPL.filePath = newArray();
	APPL.filePath = file_list('/home/reports/');

> file_mkdirs	> // creates a new directory 'sept' in '/home/reports/2004'
	APPL.mkdir = file_mkdirs('/home/reports/2004/sept');

> file_read	APPL.content = file_read('/home/reports/logfile1');
	> //start from 23rd byte
	APPL.content = file_read('c:\\batch\\myfile.txt', 23);
	> //read only 200 bytes
	APPL.content = file_read('/home/reports/logfile1', 0, 200);

> file_readLines	APPL.MyFile=new Array();
	APPL.MyFile=file_readLines('/home/reports/logfile1');
	> // start from 3rd line
	APPL.MyFile=file_readLines('c:\\batch\\myfile.txt', 3);
	> // read only first 20 lines
	APPL.MyFile=file_readLines('/home/reports/logfile1', 0, 20);

File-level Functions

Function Name	Example Code
> file_renameTo	> If (file_renameTo('/home/reports/2004/sept', '/home/reports/2004/09')) ...;
> file_replaceContents	> APPL.replace = file_replaceContents('c:\\batch\\myfile.txt', 'new file data...');

Integer Function

Function Name	Function Description
> format_toInt	> Converts a number to an integer format string. If the function is successful, it returns a string that contains the integer value. If the function is unsuccessful, the specified number is not in the correct format and "0" is returned.
	Function Syntax
	> format_toInt(number)
	Example Code

	> APPL.int = file_toInt('1.2333'); // returns '1'
	> APPL.int = file_toInt(4.0); // returns '4'

Job, subApplication, & Application control function

Function Name	Function Description
> execCommand	> Use the execCommand built-in function to control a job, a subApplication, or an Application. You can use the execCommand function at trigger time or at run time.
	Function Syntax
	> execCommand('jobname | ALL | SUBAPPL(subApplname)','applname.gen', 'ACTION the_action')
	Example Code

	> Setting user status
	execCommand('UNIX1','TEST.0','ACTION USERSTATUS TEXT("This is a test")');
	> Dropping multiple predecessors
	execCommand('UNIX2,'TEST.0','Action DropDep Pred("TASK1,TASK2")');
	> Removing a job's submission time
	execCommand('UNIX5','TEST.%APPL._gen','Action Reset Earlysub()');
	> Holding a subApplication
	execCommand('SUBAPPL(SUBC)','TEST.2','ACTION HOLD');

Alert Function

Function Name	Function Description
> WOB.isAutoResubmit	> Use the WOB.isAutoResubmit built-in function to check if a job that has failed is going to be resubmitted.
	Function Syntax
	> WOB.isAutoResubmit()
	Example Code

	> Checking if a Failed Job is Going to be Resubmitted
	> When a job fails, an Alert triggers and runs the following JavaScript script. The script triggers the CYBERMATION.VERIFY Event if the failed job is going to be resubmitted.
	
	`if(WOB.isAutoResubmit())`
	`  execTrigger('CYBERMATION.VERIFY');`

Resource function

Function Name	Function Description
> resetResourceProperty	> You can use the resetResourceProperty built-in function to set a resource's availability count. > For example, you can run a JavaScript script that automatically resets the availability count of a resource at a specific time of day.
	Function Syntax
	> resetResourceProperty('resname','property','count');
	Example Code
	> The following example sets the availability count of a resource named LOWPRIO to 1: resetResourceProperty('LOWPRIO','Availability','1');

Symbolic variable functions

Function Name	Function Description
> defined	> Use the defined function to determine if a symbolic variable is already defined. The CA WA server returns a value of true or false.
	Function Syntax
	> defined('variable')
	Example Code
	> If global variable is defined > In the following example, an Application-level script defines a symbolic variable APPL.alltrue when certain circumstances occur. If these circumstances do not occur, the variable is not defined. The job-level script is as follows: if (defined('APPL.alltrue')) WOB.truecalc = APPL.alltrue * 40;

Symbolic variable functions

Function Name	Function Description
> genTime	> Use the genTime function to create user-defined date and time symbolic variables, similar to the built-in date and time variables, for any date you choose. The genTime function creates a standard set of symbolic variables when you provide a prefix for the variable names.
	Function Syntax
	> genTime('prefix','date')
	Example Code
	> Generating symbolic variables for next workday > The following example defines a set of symbolic variables for the next workday, beginning with the characters NW: genTime('NW','today plus 1 workday');

Symbolic Variables

- Symbolic variables provide powerful substitution capabilities.
- In a JavaScript script, you can define your own symbolic variables or use one of the CA WA server's built-in symbolic variables.
- When the server encounters a symbolic variable in an Application, job or Alert, it substitutes the current value of that variable.
- You can use symbolic variables to define date parameters, specify job names, pass arguments to scripts, and many more functions.
- You can export the value of a symbolic variable to one or more jobs in an Application or to the Application itself.

Identifying symbolic variables

> Within the JavaScript script, symbolic variables are identified by their prefix (ESP, APPL or WOB), which allows their values to be exported from the script.

> Within an Application or job, symbolic variables are identified by their symbol introducer character.

> A symbolic variable begins with the percent sign (%) and ends with a space, another symbolic variable introducer character, a character that is not valid in a symbolic variable such as an asterisk (*), or any other character that is not alphanumeric or an underscore.

Identifying symbolic variables

- A symbolic variable is defined as a variable within a JavaScript script. As such, it must conform to the rules of JavaScript variable definitions.
- Each symbolic variable must have a prefix (ESP, APPL or WOB) that identifies the CA WA server host object it belongs to.
- If it is not part of a host object, the value of a symbolic variable cannot be exported from the script, and you will not be able to use the variable in a field.
- A symbolic variable name may contain alphanumeric characters and underscores. The first character cannot be numeric. The name of the symbolic variable can be as long as you want; however, it is best to keep symbolic variable names to a manageable length.

Identifying symbolic variables

- A symbolic variable may be assigned a text string or a numeric value, such as the following:

  ```
  ESP.DC='Toronto';
  ESP.NUM=78;
  ```

- When you assign a symbolic variable a value that is a text string, you must enclose the string in single or double quotation marks.
- **Note**: The following names are reserved for server built-in symbolic variables:

  ESP._*name*

  APPL._*name*

  WOB._*name*

Availability of symbolic variable values

> The resolved value of a symbolic variable is made available to jobs and Applications as follows:
  - At the system level, where its value is available to any Application. This symbolic variable is identified by the prefix **ESP**.
  - At the Application level, where its value is available to any job within the Application. This symbolic variable is identified by the prefix **APPL**.
  - At the job level, where its value is confined to the specific job. This symbolic variable is identified by the prefix **WOB**.

> **Note**: When you define a JavaScript variable, you must identify the scope of its availability if you want the value of the variable to be available outside of the script. Variables with no prefix cannot be referenced outside the script.

Using built-in symbolic variables

- The CA WA server provides several predefined symbolic variables that you can use as required. You do not need to define these variables to use them. You can use these variables in two ways:
  - Within a JavaScript script when you want the value of the variable for use within the script—simply specify the variable with the appropriate prefix within the script.
  - Within CA WA Desktop Client when you want to specify the resolved value of the variable directly in an input field—simply include the variable preceded by its symbol introducer wherever you need to use the value the symbolic variable resolves to.
- The server built-in symbolic variables consist of two parts: a prefix that identifies the scope of the symbolic variable and the symbolic variable name, separated by a period.

APPL-prefixed built-in symbolic variables

Variable Name	Description
> APPL._alert	> This variable is only available in Alerts. Use %APPL._alert when you need the name of the Alert. The variable resolves to the Alert name if one is specified; otherwise, it resolves to null. The value is in uppercase.
> APPL._connectionfactory	> Use %APPL._connectionfactory when you need the name of the connection factory used in a JMS Subscribe Event. The connection factory contains all the bindings the CA WA server needs to look up the referenced topic or queue.
> APPL._destination	> Use %APPL._destination when you need the name of the destination file used in a JMS Subscribe Event. The destination file stores messages consumed from a topic or queue.
> APPL._dsn	> Use %APPL._dsn when you need the name of the data set monitored in a z/OS Data Set Trigger Event.
> APPL._event	> Use %APPL._event when you need the full name of the Event. The Event name resolves in uppercase, for example CYBER.PAYROLL.

APPL-prefixed built-in symbolic variables

Variable Name	Description
> APPL._eventid	> Use %APPL._eventid when you need the name of the SAP event monitored in a SAP Monitor Event.
> APPL._eventname	> Use %APPL._eventname when you need the name of the Event without the prefix. The Event name resolves in uppercase. For example, suppose that the full name of an Event is CYBER.PAYROLL. The %APPL._eventname variable resolves to PAYROLL.
> APPL._eventparam	> Use %APPL._eventparam when you need the name of the SAP event parameter, such as a job name or job count, used in a SAP Monitor Event.
> APPL._eventprefix	> Use %APPL._eventprefix when you need the prefix of the Event name. The Event prefix resolves in uppercase. For example, suppose that the full name of an Event is CYBER.PAYROLL. The %APPL._eventprefix variable resolves to CYBER.
> APPL._initialcontextfactory	> Use %APPL._initialcontextfactory when you need the name of the initial context factory used in a JMS Subscribe Event. The initial context factory acquires an arbitrary initial context that the application can used and is required within the JNDI framework.

APPL-prefixed built-in symbolic variables

Variable Name	Description
> APPL._filename	> Use %APPL._filename when you need the name of the file monitored in a File Trigger Event.
> APPL._ftfile	> Use %APPL._ftfile when you need the name of the file monitored in a file trigger job. The variable resolves to the last file trigger that occurred within the Application. For example, you can use this variable in a successor job to process a file that caused the file trigger to occur.
> APPL._gen	> Use %APPL._gen when you need the absolute generation number of the Application, for example 24.
> APPL._jndidestination	> Use %APPL._jndidestination when you need the JNDI name of the topic or queue monitored in a JMS Subscribe Event.
> APPL._loadmode	> Use %APPL._loadmode to determine if an Application is running. The variable returns a string that resolves to "false" if the Application is running and resolves to "true" otherwise.

APPL-prefixed built-in symbolic variables

Variable Name	Description
> APPL._name	> Use %APPL._name when you need the name of the Application. The value is in uppercase, for example PAYROLL.
> APPL._providerurl	> Use %APPL._providerurl when you need the URL of the JMS Provider used in a JMS Subscribe Event.
> APPL._rootjobs	> Use %APPL._rootjobs when you need the names of the root jobs specified in an Event trigger. The %APPL._rootjobs variable resolves to the root job names including any unresolved symbolic variables in the names. If no root jobs are specified, the %APPL._rootjobs variable resolves to an empty string.
> APPL._tag	> Use %APPL._tag when you need the Application tag. The variable resolves to the Application tag if one is specified; otherwise, it resolves to null. The value is in uppercase.
> APPL._truser	> Use %APPL._truser when you need the user ID that triggered this Event. For scheduled and monitor Events, this is the Event's execution user (which defaults to the user that created or last modified the Event). The value is in uppercase, for example OPER1.

APPL-prefixed built-in symbolic variables

Variable Name	Description
> APPL._user	> Use %APPL._user when you need the user ID of the last person to create or edit the Event. The value is in uppercase, for example SCHEDMASTER.
> APPL._user1 through APPL._user4	> Use %APPL._usern when you need to pass user parameters to an Event when it is triggered or simulated. Specify the value for n that corresponds to the user parameter you will use when triggering or simulating. The case you specify the parameters in is preserved, where required.

WOB-prefixed built-in symbolic variables

Variable Name	Description
> WOB._agent	> Use the %WOB._agent symbolic variable when you need the name of the agent the job ran on. You can use this symbolic variable in a JavaScript script from an Application or an Alert.
> WOB._avgruntime	> Use %WOB._avgruntime when you need the average run time for a job. By default, the CA WA server uses the last ten executions of the job to calculate the average run time.
> WOB._cmpc	> Use %WOB._cmpc when you need the completion code of the job. The value of this variable is available when a job completes or fails.
> WOB._fullname	> Use %WOB._fullname when you need the full name of the job, including the qualifier. The name resolves in uppercase, for example PAYJOB1.RUN1. If the qualifier is not specified, the %WOB._fullname variable resolves to the job name.
> WOB._jobno	> This variable applies to Alerts only. Use %WOB._jobno when you need the job number of the job that executes the Alert. It is an integer variable and is set to 0 for a job that has not been submitted.

WOB-prefixed built-in symbolic variables

Variable Name	Description
> WOB._lstatus	> Use %WOB._lstatus when you need the long status of the job. The long status provides additional information about the job. If the job does not have a long status, the %WOB._lstatus variable resolves to null.
> WOB._name	> Use %WOB._name when you need the name of the job, excluding the qualifier. The name resolves in uppercase, for example PAYJOB1.
> WOB._qualifier	> Use %WOB._qualifier when you need the qualifier of the job name. The value resolves in uppercase, for example RUN1.
> WOB._retrycount	> Use %WOB._retrycount when you need the retry count of the job. The retry count specifies the number of times the CA WA server tries to resubmit a job if it fails.
> WOB._state	> This variable applies to Alerts only. Use %WOB._state when you need the value of the job's state.

WOB-prefixed built-in symbolic variables

Variable Name	Description
> WOB._status	> Use %WOB._status when you need the job status sent by the agent or CA WA server. You can use this symbolic variable in a JavaScript script from an Application or an Alert.
> WOB._subcount	> Use %WOB._subcount when you need the job submission count. The count is incremented each time the job is submitted. If the job has not been submitted, the %WOB._subcount resolves to 0 (zero).
> WOB._tag	> Use %WOB._tag when you need the value for the job tag. The variable resolves to the job tag if it is specified in the job definition; otherwise, it resolves to the Application tag if one is specified. If neither tag is specified, the variable resolves to null. The value is in upper case.
> WOB._type	> Use %WOB._type when you need the type of job.
> WOB._userstatus	> Use %WOB._userstatus when you need the user status of a job. You can use this symbolic variable in a JavaScript script from an Application or an Alert.

Date and time built-in symbolic variables

- The CA WA server provides several built-in date and time symbolic variables that you can use for scheduled, actual or runtime dates or times.
- The first character after the period determines if it is a scheduled (S), actual (A,) or runtime (R) date or time.
- For example, the scheduled date symbolic variable is APPL._SDATE. The corresponding actual date symbolic variable is APPL._ADATE, and the corresponding runtime date symbolic variable is APPL._RDATE.
- Scheduled dates and times are based on the time the Event is scheduled to trigger or, if the Event is manually triggered, the trigger time for the Event.
- Actual dates and times are based on the actual system time when the Application is generated.

Date and time built-in symbolic variables

Date/Time Variable	Description
APPL._SDATE APPL._ADATE APPL._RDATE WOB._RDATE WOB._LDATE	> Date in full Example: Friday 31st March 2009. If you want to pass this as an argument, you need to enclose it in double quotation marks. > For example: "%APPL._SDATE"
APPL._SYY APPL._AYY APPL._RYY WOB._RYY WOB._LYY	> Last two digits of the year > Example: 09
APPL._SYEAR APPL._AYEAR APPL._RYEAR WOB._RYEAR WOB._LYEAR	> Year > Example: 2009

Date and time built-in symbolic variables

Date/Time Variable	Description
APPL._SMM APPL._AMM APPL._RMM WOB._RMM WOB._LMM	> Number of month > Example: 10 for October
APPL._SMMM APPL._AMMM APPL._RMMM WOB._RMMM WOB._LMMM	> First three characters of month > Example: Oct
APPL._SMONTH APPL._AMONTH APPL._RMONTH WOB._RMONTH WOB._LMONTH	> Name of month > Example: October

Date and time built-in symbolic variables

Date/Time Variable	Description
APPL._SDAY APPL._ADAY APPL._RDAY WOB._RDAY WOB._LDAY	> Name of day > Example: Monday
APPL._SDD APPL._ADD APPL._RDD WOB._RDD WOB._LDD	> Number of actual day of month > Example: 09
APPL._SDDD APPL._ADDD APPL._RDDD WOB._RDDD WOB._LDDD	> Julian day, or the number of the day in the year > Example: 045

Date and time built-in symbolic variables

Date/Time Variable	Description
APPL._SDOWNUM APPL._ADOWNUM APPL._RDOWNUM WOB._RDOWNUM WOB._LDOWNUM	> Number of day in week > Example: 1 for Sunday, 2 for Monday, and so on.
APPL._SDOW# APPL._ADOW# APPL._RDOW# WOB._RDOW# WOB._LDOW#	> Number of actual day of month > Example: 09
APPL._STIME APPL._ATIME APPL._RTIME WOB._RTIME WOB._LTIME	> Time in 24-hour format > Example: 14.55.32

Date and time built-in symbolic variables

Date/Time Variable	Description
APPL._SHH APPL._AHH APPL._RHH WOB._RHH WOB._LHH	> Hour in 24-hour format > Example: 08
APPL._SMN APPL._AMN APPL._RMN WOB._RMN WOB._LMN	> Minute of hour > Example: 55
APPL._SSS APPL._ASS APPL._RSS WOB._RSS WOB._LSS	> Number of seconds past the minute > Example: 32

Defining a system-level symbolic variable

> You can define your own system-level symbolic variables if you require a variable that is not provided. You can use a system-level symbolic variable in any Application or job running on the CA WA server. System-level symbolic variables are stored in the ESP host object. You can define a system-level symbolic variable in the following ways:
>
> - Define the variable as a default in the Resources.Define.UserESP file
> - Define the variable in a JavaScript script

> All system-level symbolic variables start with the ESP prefix.

> **Note:** After a cold start of the CA WA server, system-level variables that are defined in a JavaScript script are cleared or, if a default exists, restored to their default value. To preserve a system-level variable and its value, define it as a default instead.

Global variables

- Global variables let you store information that you can reuse across Applications. Global variables save time: you do not have to enter specific information, such as job names or argument values, each time you want to perform the same kind of processing. When you use global variables, you also reduce the possibility of coding errors.
- Global variables are stored in the relational database for CA WA. Each global variable belongs to a context, which is a group of related variables. Contexts help you avoid naming conflicts.
- You can use global variables when you define jobs. The %VAR statement lets you specify a global variable name in supported job definition fields.
- When an Event is triggered, the CA WA server substitutes the current value of that global variable. You can also define jobs that have a dependency on global variables. The job is submitted after all of the job's dependencies (time, predecessor, variable, and resource dependencies) are met.

Global variables

- Although both global variables and system-level symbolic variables let you store information that you can reuse across Applications, they are created and managed differently.
- A symbolic variable is a JavaScript variable whose value can be accessed outside the context of the JavaScript script.
- All symbolic variables are stored in built-in JavaScript host objects.
- Unlike symbolic variables, global variables are not dependent on JavaScript. Instead, global variables are stored in the relational database for CA WA.
- You can specify both types of variables in supported job definition fields and use them in JavaScript scripts.
- When CA WA encounters a symbolic variable or the global variable %VAR statement in a job definition field, it substitutes the current value of that variable.

Global variables

> Another difference between the two types of variables is that global variables support variable dependencies.

> Whereas symbolic variables only let you substitute values in job definition fields, global variables let you define jobs that run when their global variable expressions are satisfied.

> For example, you can define a job that only runs when a global variable named quota is assigned a value greater than or equal to 1000.

%VAR Statement—Specify a global variable in a job definition

> You can use global variables when you define and monitor jobs.

> The %VAR statement lets you specify a global variable name in supported job definition fields in the Define and Monitor perspectives of CA WA Desktop Client.

> When an Event is triggered, the CA WA server tries to substitute the current value of a global variable specified in the %VAR statement.

> Example:

```
WOB.locationvalue = %VAR(‘location1’,’dclocations’)
```

JavaScript Examples

> Overriding average runtime for estimated end times

- For example, if you know that the job normally takes 10 times longer to execute on a Friday, use the following script in the Application:

```
if (today('friday')) WOB.duration=WOB._avgruntime * 10;
  else WOB.duration=WOB._avgruntime;
```

- Run this script at Event trigger time, so that the server calculates the correct value.
- To complete this example, specify the %WOB.duration symbolic variable in the Duration in minutes field on the job's General dialog.

JavaScript Examples

> Scheduling a job to run based on day and time

- An Event is scheduled every hour on the hour to run an Application. Job X in the Application should only be selected to run on Fridays at 3 p.m.
  - To schedule a job to run based on day and time
  1. Use the following JavaScript script at Event trigger time for job X:
     ```
     if (today('friday') && APPL._SHH=='15')
        WOB.runme='true';
     else WOB.runme='false';
     ```
  2. Use the %WOB.runme variable as the run frequency in the job definition.
  3. Schedule an hourly Event to run the Application.
- Job X is selected to run only if it is Friday and the Event's scheduled hour (APPL._SHH) is 15 (in other words, the Event was scheduled at 3 p.m.).

JavaScript Examples

> Running different jobs based on the scheduled hour

- An Event is scheduled every hour on the hour. Different jobs run based on the scheduled hour. The following is the schedule frequency for each job:
  - JOBA runs at 13:00, 15:00, and 17:00.
  - JOBC runs at 08:00, 12:00, and 16:00.
  - All other jobs run each hour.

JavaScript Examples

> Running different jobs based on the scheduled hour

1. Use the following JavaScript script at Event trigger time for each job:

```
WOB.runme=false;
//run different jobs based on scheduled hour
switch (WOB._name)
{
  case 'JOBA':
      if (APPL._SHH == '13' || APPL._SHH == '15' ||
       APPL._SHH == '17') WOB.runme=true;
      break;
  case 'JOBC':
      if (APPL._SHH == '08' || APPL._SHH == '12' ||
       APPL._SHH == '16') WOB.runme=true;
      break;
  default:
  WOB.runme=true;
}
```

2. Use the %WOB.runme variable as the run criteria for each job.

- The WOB.runme variable is set based on the job name and the scheduled hour. “case” statements are used for jobs with special requirements. If the name of a job in the Application does not match one of the case statement labels, WOB.runme is set to true for the job.

JavaScript Examples

> Scheduling a different script each quarter

- Schedule a job and run a different script for the job each calendar quarter. Each calendar quarter consists of three months, and the first quarter starts on January 1.
  1. Use the following JavaScript script at run time for the job:
     ```
     if (today('jan feb mar')) WOB.scriptname='quarter1';
     if (today('apr may jun')) WOB.scriptname='quarter2';
     if (today('jul aug sep')) WOB.scriptname='quarter3';
     if (today('oct nov dec')) WOB.scriptname='quarter4';
     ```
  2. Use the %WOB.scriptname variable in the Script/command name field for the job, for example:
     ```
     /export/home/jsmith/%WOB.scriptname
     ```
- In a single job definition, the JavaScript script determines the current calendar quarter, and sets the WOB.scriptname variable to “quarter1”, “quarter2”, “quarter3”, or “quarter4”.
- When the job is ready to run, the CA WA server resolves this variable and runs the appropriate script.

JavaScript Examples

> Creating multiple dynamic executions of the same job with different arguments per execution

  - An jobstream needs to run a script and extract information for arguments for each run of the same job. The number of unique arguments fluctuates from day to day. So the number of jobs needed to process all the unique argument mixes is not static. All unique parameter sets are loaded in to a CA Workload Automation DE Global Variable Context with all variables starting with DAVE and appended to that a number for each unique parameter set.

JavaScript Examples

```
var counter = getVar('DAVECOUNTER','TEST');
var x = 1;
while (x<=counter) {
    var parm = getVar('DAVE' + x, 'TEST');
    var name = parm.split(" ");
    execCommand( "DATA_STAGE" + '.' + name[0], APPL._name + '.' +
    APPL._gen, "ACTION Insert Type(UNIX) Container(%(APPL._name)~~)
    Rununit(DW005) Tag(" + '\"' + parm + '\"' + ")
    Command(/export/home/dsadm/Ascential/DataStage/DSEngine/bin/dsjob
    ) Pred(START.APPL) Succ(STOP.APPL) User(dsadm) Args(" + '\"' + parm
    + '\"' + ")" );
    x++;
}

execCommand('STOP.APPL', APPL._name + '.' + APPL._gen, 'ACTION
    RELEASE');
```

JavaScript Examples

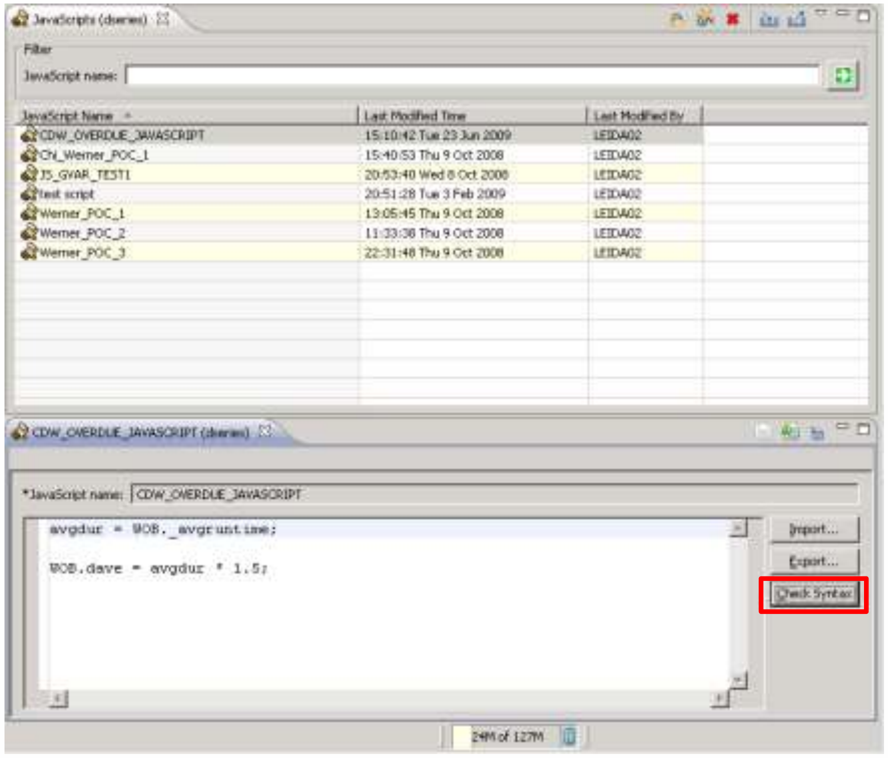
> Creating multiple dynamic executions of the same job with different arguments per execution

- The JavaScript is defined as part of link workload object definition to execute at link run time, reads the number of parameter sets in the context from a control variable (DAVECOUNTER) and then inserts a job per parameter set.

Troubleshooting

> JavaScript Syntax Checker

- Check Syntax button when editing JavaScripts checks format and provides feedback on statement errors
- Works with JavaScripts stored in the Java Script Repository



JavaScript Name	Last Modified Time	Last Modified By
CDW_OVERDUE_JAVASCRIPT	15:10:42 Tue 23 Jun 2009	LEIDA02
CN_Werner_POC_1	15:40:53 Thu 9 Oct 2008	LEIDA02
JS_GVAR_TEST1	20:53:40 Wed 8 Oct 2008	LEIDA02
test script	20:51:28 Tue 3 Feb 2009	LEIDA02
Werner_POC_1	13:05:45 Thu 9 Oct 2008	LEIDA02
Werner_POC_2	11:33:38 Thu 9 Oct 2008	LEIDA02
Werner_POC_3	22:31:48 Thu 9 Oct 2008	LEIDA02

*JavaScript name: CDW_OVERDUE_JAVASCRIPT

```
avgdur = WOB._avgruntime;

WOB.dave = avgdur * 1.5;
```

Questions?