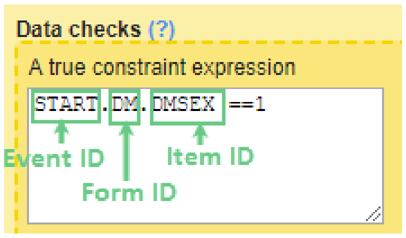
VIEDOC eLearning

Using JavaScript in Viedoc

1 Introduction

- In Viedoc Designer, JavaScript can be used to provide a lot more flexibility when working with the study design, by:
  - setting **Visibility** conditions (for items, item groups, activities and events).
  - writing data checks under the **Validation** tab for items.
  - using **functions/default values** for items.
  - writing conditions for **Alerts**.
- The syntax used is the JavaScript syntax, as described in ECMAScript 5.1 standards.
- The variables used are the items in the design, referred to as described in section *Addressing Viedoc items*.
- The operators used are listed in section *Operators*.

2 Addressing Viedoc items



```
START.DM.DMSEX ==1
```

- An item from a different form (a cross form item) is addressed in Viedoc in the following format: *Event.Form.Item*. It consists of three identifiers separated by ".":
  - The event identifier - see *Specifying the event* below.
  - The form identifier - see *Specifying the form* below.
  - The item identifier - see *Specifying the item* below.
- **Specifying the event**

  The event can be specified in one of the following ways:
  - `EventID` - the Event ID as specified in the Event settings.
  - `EventID[StudyEventRepeatKey]` - if the event is recurring and you want to specify a certain occurrence, this is done by writing its `StudyEventRepeatKey` value within [ ].
  - by using the indexers, for example `EventID$FIRST`*n* - to identify the *n*th initiated event, in case the specified form appears in multiple events. See *Relative path* section below for a complete description of the indexers.
- **Specifying the form**

  The form can be specified in one of the following ways:
  - `FormID` - the form ID as specified in Viedoc Designer under form settings.
  - `FormID[FormRepeatKey]` - if the form is a repeating form and you want to specify a particular instance, this is done by writing its `FormRepeatKey` value within [ ].
  - `FormID$ActivityID` - optionally, it is possible to specify the activity, in case the respective form appears in multiple activities within the same event.

    E.g. `DM$MORNING.WEIGHT` - refers to the WEIGHT item in the DM form within the MORNING activity.
  - `FormID$ActivityID[FormRepeatKey]` - if you want to specify both the activity and a particular instance of a repeating form, this is specified in this format.
- **Specifying the item**

  To access an item in the same form, it is sufficient to use the `ItemID` only, there is no need to specify the event and form. In this case, when changing the value of the input item in Viedoc Clinic, the result item (depending on the input item) will update its value at the same time.

In case of using the *Event.Form.Item* for an item within the same form, when changing the value of the input item in Viedoc Clinic, the result item value will be updated only after saving the form and re-opening it.

To access an item in another form (so called cross form item) it is necessary to specify the event and the form.

```
EventID.FormID.ItemID
```

- **Notes!**

• An event is any event defined in workflow.
• The JavaScript language definition allows space between keywords, but spaces inside cross form variable are not allowed, because, in order to provide values to execution context, Viedoc parses the whole expression for syntax as specified above.

- **Relative path**

It is also possible to access cross form variable using relative path. Some special keywords are used in this case. Keywords that can be appended to `StudyEventDefId`. These keywords can also be used without `StudyEventDefId` to select any event.

`$FIRST`*n*	Select the first event with the form initiated, *n* is an optional index and it goes forward (meaning that on a timeline `$FIRST2` comes before `$FIRST3`)	`AE$FIRST.AEFORM.AEDATE`
		`$FIRST.DM.HEIGHT`
		`$FIRST2.DM.WEIGHT`

`$LAST`*n*	Select the last event with the form initiated, *n* is an optional index and it goes backwards (meaning that on a timeline `$LAST2` comes after `$LAST3`)	`AE$LAST.AEFORM.AEDATE`
`$PREV`*n*	Select the previous event with the form initiated, *n* is an optional index and it goes backwards (meaning that on a timeline `$PREV2` comes after `$PREV3`	`WEIGHT < $PREV.DM.WEIGHT + 10`

In addition

`$THIS`	Select the current context event

Here comes an example of using the optional indexer (*n* in the above table):
On the *Add patient* form, we have a text item that should have the latest non-blank value of another text item (called *NAME*) that is present on a form (called *PROFILE*) which is present on all scheduled (the first one called *START*) and unscheduled events. The function below can be used on the item of the *Add patient* form:

```
if($LAST.PROFILE.NAME != null) return $LAST.PROFILE.NAME;
if($LAST2.PROFILE.NAME != null) return $LAST2.PROFILE.NAME;
if($LAST3.PROFILE.NAME != null) return $LAST3.PROFILE.NAME;
if($LAST4.PROFILE.NAME != null) return $LAST4.PROFILE.NAME;
if($LAST5.PROFILE.NAME != null) return $LAST5.PROFILE.NAME;
if(START.PROFILE.NAME != null) return START.PROFILE.NAME;
return 'NOT SET';
```

This will allow for the item being saved blank for the last 4 events, or else fallback to the value of the item on the start event.

- **Cross Event Date**

Accessing the date of another event uses same principle as any cross form variable, use a fixed form id `$EVENT` and item id `EventDate`.

`AESTDT >= BL.$EVENT.EventDate`	Start date must be after BL visit date

3 Operators

- The below operators are common among other programming languages so if you have any programming experience, you will be well at ease with the operators in JavaScript.
- Arithmetic Operators:

  • + (addition)
  • - (subtraction)
  • * (multiplication)
  • / (division)
- Comparison:

  • == (equal to)
  • != (not equal to)
  • < (less than)
  • > (greater than)
  • <= (less than or equal)
  • >= (greater than or equal)
- Boolean logic:

  • && (AND)
  • || (OR)
  • ! (NOT)

4 Data types

- The following table lists the Viedoc items together with their JavaScript types and default values.

Viedoc item	**JavaScript type**	**Default value**
Single line text	String	null
Number	Number	null
Date	Date object	null
Time	Date object	null
Paragraph text	String	null
Checkbox	Array of string/number*	[ ]
Radio button	String/Number*	null
Dropdown	String/Number*	null
File	Object with following members: ◦ FileName (string) - name of the uploaded file ◦ FileSize (number) - file size in bytes ◦ FileHash (string) - MD5 hash of the file content	null
Range	String representation of a range object (see more details in section *Viedoc provided functions > Range item specific functions and properties*).	null

**The item type for checkbox, radio button or dropdown is usually number, unless any of the choice codes is not a number.*

5 Pass by value vs. pass by reference

- In JavaScript data is passed in two ways: by value and by reference, respectively.

- The following JavaScript data types are the ones that are "passed by value": Boolean, null, undefined, String, and Number.

  This means that, when comparing two variables of the above mentioned types, their values will be compared. It means that, for example if we have the following code:

  ```
  var a = 3;
  var b = 'def';
  var x = a;
  var y = b;
  ```

  ...then a will get the value 3, b will get the value 'def', x will get the value 3, y will get the value 'def'. Variables a and x have the same value (3), and b and y have the same value ('def'). Still, all the four variables are independent. If we continue now and change the value of a to 5, this will have no impact on var x, which still has the value of 3.

- The following JavaScript data types are "passed by reference": Array, Function, and Object.
  This means that variables of these types get a reference to a certain value, not the value itself.
  See *Data types* section for information on which Viedoc items are objects.

  The date is always an object in JavaScript. If we have for example, the following:

  ```
  var d=new Date();
  var c=d;
  d.setDate(10);
  ```

  ...then variable d is created as a date object, variable c is assigned to reference to the same value as d, and then each and every time we change d, c will dynamically change as well, as it references the same value as d does. So, when we set d to the 10th of the current month, c will automatically get the same value.

6 System variables

- When an expression is evaluated in a form context, following variables are accessible.

Variable name	Data type	Default value
`{ItemDefId}`	As specified in data types table in section 4	As specified in data types table in section 4
`SubjectKey`	String	null only for add patient
`SiteSubjectSeqNo`	Number	Sequence number of the subject in the site (starts with 1)
`StudySubjectSeqNo`	Number	Sequence number of the subject in the study (starts with 1)
`SiteCode`	String	The site code as set in Viedoc Admin
`CountryCode`	String	Two letter ISO country code of the site
`StudyEventDefId`	String	The ID of the study event as specified in the study workflow in Viedoc Designer
`StudyEventType`	String	"Scheduled", "Unscheduled" or "Common"
`StudyEventRepeatKey`	String	The number that identifies a specific recurrence of a recurring event
`FormDefId`	String	The ID of the form as specified in Viedoc Designer > forms > settings
`FormRepeatKey`	String	The number that identifies the specific instance of a repeating form within a specific activity
`EventDate`	Date object	Current date for common events, all other current events date
`ActivityDefId`	String	The ID of the activity as specified in the study workflow in Viedoc Designer
`{ItemDefId}__format`	Number	Date types:

0. Date only
1. Date and time
2. Day not known
3. Month not known

e.g. `ICDATE_format`

Numeric with decimals
`n:` precision/number of decimals

See also [Controlling the format of a date/time variable]().

7 Expressions

- An expression in Viedoc is a JavaScript function body written in ECMAScript 5.1 standards http://www.w3schools.com/js/js_syntax.asp

  E.g.

  ```
  return 2;
  ```

  ```
  var a=2;
  var b=3;
  return a+b;
  ```

- Expressions can be written as **single statement**. Viedoc converts them into function body.

  `expression` is converted to `return (`*`expression`*`)`:

  ```
  2
  ```

  converts to

  ```
  return (2);
  ```

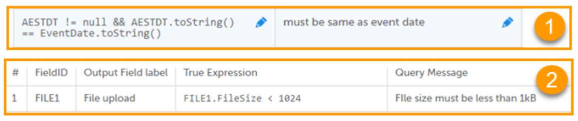
  ```
  VSDT <= now()
  ```

  converts to

  ```
  return ( VSDT <= now());
  ```

- **WARNING!** When using loops (`for, for/in, while, do/while`) consider the following:
  - Do not copy code (from internet) without reviewing it first and checking what it does.
  - Make sure that you don't have endless loops.
  - Consider browser compatibility when using javascript functions.
  - Avoid inline/recursive functions, as they could cause memory problems in the system, both on the client as well as on the server side.

8 Boolean expressions



```
AESTDT != null && AESTDT.toString()
== EventDate.toString()
```

must be same as event date

#	FieldID	Output Field label	True Expression	Query Message
1	FILE1	File upload	`FILE1.FileSize < 1024`	FIle size must be less than 1kB

- The boolean expressions are the expressions that return a boolean value, either `true` or `false`. They are used in the **Visibility** and **Validation**.

  For JavaScript comparators and logical operators, you can refer to http://www.w3schools.com/js/js_comparisons.asp.

  - `true` - everything with a real value is `true`:

    ```
    100
    ```

`"Hello"`

`"false"`

`7 + 1 + 3.14`

`5 < 6`

- `false` - everything without a real value is `false`:

  `0`

  `Undefined`

  `null`

  `""`

- E.g. A validation expression to check that the weight is > 65 for males ('M') and > 45 for females. Gender (ItemID=`GENDER`) is collected in Patient Information form (FormID = `PI`) withing the Screening event (EventID = `SCR`) and weight (ItemID = `WEIGHT`) is collected in Demographics form in each visit. Below is the edit check written for the `WEIGHT` item:

```
if (SCR.PI.GENDER == 'M')
return WEIGHT > 65;
else
return WEIGHT > 45;
```

- **Date comparison**

  Dates in JavaScript are objects and the available comparators are > , >= ,
  < and <=.

  If you need to use == or !=, convert to string and compare, as in example (1) in the image, except for the case when checking for NULL. When checking for NULL, do not convert as this would lead to a change of the value during conversion from NULL to something different.

  Note! Remember to check for NULL before invoking any function on an object.

  See the *Pass by value vs. pass by reference* section for an explanation on working with objects.

- **File properties**

  The metadata values of the file datatype can be accessed in expressions, as shown in the image at (2).

9 Default value

- The default value expressions are executed and the resulting value is set only when the form is initialized.

  Note! If there is visibility condition set on item with function or default value, whenever item becomes hidden, their value is reset to default value.

10 Viedoc provided functions

- **`date(dateString)`**

  Converts date string to JavaScript date object. The dateString must be in "yyyy-mm-dd" format.

- **`today()`**

  Returns current site date.

- **`now()`**

Returns current site date and time (in site timezone).

- **`addDays (date, days)`**

  Add the specified number of days to the specified date object.

- **`age (fromDate, toDate)`**

  Returns the age of the subject based on the provided input dates, for example `fromDate` = date of birth and `toDate` = date of informed consent.

  The function implementation is shown below:

```
function age(DMDOB, DMIC) {
   if (!DMIC || !DMDOB)
      return null;

   var ageMilli = DMIC - DMDOB;

   var ag = ageMilli / 1000 / 3600 / 24 / 365.25;

   return ag;
}
```

- **`bmi ( weightInKg, heightInCM )`**

  Returns the bmi (body mass index) calculated based on the provided weight (in kg) and height (in cm). The function implementation is shown below:

```
function bmi(weight, height) {
   if (weight <= 0 || height <= 0)
      return null;
   var finalBmi = weight / (height / 100 * height / 100);

   return finalBmi;
}
```

- **`days ( DateA, DateB )`**

  Calculates the number of days between 2 dates provided as input parameters: `DateA` and `DateB`, as DateA - DateB.

  Note! Please note that the result is always rounded to the nearest integer (see function implementation below), and when using at least one input parameter that contains both date and time, this means that, for example, for a difference of 1.3 days the function will return 1, and for a difference of 1.7 days the function will return 2.

  The function implementation is shown below:

```
function days(endDate, startDate) {
   if (!startDate || !endDate)
      return null;

   var oneDay = 24 * 60 * 60 * 1000; //
hours*minutes*seconds*milliseconds

   startDate = date(startDate);
   endDate = date(endDate);

   var diffDays = Math.round((endDate.getTime() - startDate.getTime()) / (oneDay));

   return diffDays;
}
```

- **`hours( DateTimeA, DateTimeB )`**

  Calculates the number of hours between 2 "date and time" items provided as input parameters: `DateTimeA` and `DateTimeB`, as DateTimeA - DateTimeB.

  The function implementation is shown below:

```
function hours(endDateTime, startDateTime) {
   if (!startDateTime || !endDateTime);
      return null;

   var oneHour = 60 * 60 * 1000; // minutes*seconds*milliseconds

   var diffHours = Math.round((endDateTime.getTime() - startDateTime.getTime()) /
(oneHour));

   return diffHours;
}
```

- **`minutes( DateTimeA, DateTimeB )`**

  Calculates the number of minutes between 2 "date and time" items provided as input parameters: `DateTimeA` and `DateTimeB`, as DateTimeA - DateTimeB.

  The function implementation is shown below:

```
function minutes(endDateTime, startDateTime) {
   if (!startDateTime || !endDateTime);
      return null;

   var oneMinute = 60 * 1000; // seconds*milliseconds

   var diffMinutes = Math.round((endDateTime.getTime() - startDateTime.getTime()) /
(oneMinute));

   return diffMinutes;
}
```

- ***Array.contains* function**

  `[].contains( x )` is a special function used to check if an item is present in array.

  Examples:

  - With checkboxes:

    ```
    INCL.contains(1) || EXCL.contains(2)
    ```

  - Skip validation in all 3 activities:

    ```
    var skipActivities = ['V1A1', 'V2A1', 'V3A1'];

    if ( skipActivities.contains( ActivityDefId ) )
       return true;;
    ...
    ```

- **Range item specific functions and properties**

  The range item is the string representation of a range object. The functions that can be used to convert the respective string to a range object and vice versa are described below.The properties available for the range object are:

`RangeObject*``.Lower`	- the lower limit of the range	- number.
`RangeObject*``.LowerFormat`	- the number of decimals used for the lower limit of the range	- number.
`RangeObject*``.Upper`	- the upper limit of the range	- number.
`RangeObject*``.UpperFormat`	- the number of decimals used for the upper limit of the range	- number.
`RangeObject*``.Comparator`	- the comparator used to define the range. The available comparators are: • `InclusiveInBetween` - defines a range beween a lower and an upper defined limits. • `LessThan` • `LessThanOrEqualTo`	- string.

	• `GreaterThan` • `GreaterThanOrEqualTo` • `EqualTo` **Note!** When using the comparator in functions, make sure to write it between quotes, e.g. `"LessThan"`. It is case sensitive, so make sure to type it in exactly as stated above.	
`RangeObject*``.StringValue`	- the string representation of the respective range item	- string.

**RangeObject* can be obtained as output of the first two functions described below.

The functions available to be used in conjunction with the range item are described in the table below.

The `inRange` function can be used to verify if a certain value of a numeric item is included in a specified range. The numeric value is specified by the numeric item ID and the range is specified either by the range item ID or by the string representation of the respective range item.

Function	Input	Output	Example
`parseRangeValue (value)`	value as string	range object, null if the input is empty or if it cannot be parsed.	• `parseRangeValue("[1.3,2.0]")` • `parseRangeValue (`*`range_item_ID`*`)`
`createRangeValue (lower, comparator,upper)`	lower limit as string or float, comparator as string, upper limit as string or float	range object, null if the input is empty or if it cannot be parsed.	`createRangeValue("1.3", "InclusiveInBetween", "2.0")`

`getRangeValue (rangeValue)`	rangeValue as range object	string respresentation of the range defined by the input range object	`function() {`
			`var rangeValue = createRangeValue("1.3", "InclusiveInBetween", "2.0");`
			`return getRangeValue (rangeValue);`
			`}`

`inRange (rangeValue, numericValue)`	range value as string or range object, numeric value	• boolean value showing if the input numeric value is within the input range (true) or not (false). • true if at least one of the input parameters is missing. • false if one of the input parameters is invalid.	• `inRange(`*`range_item_ID`*`, `*`numeric_item_ID`*`)` • `inRange("[1,3]", `*`numeric_item_ID`*`)`

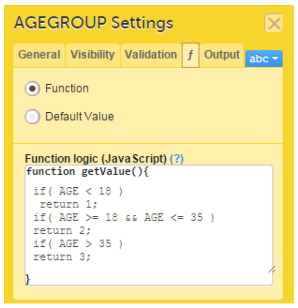
11 Math library

- The ECMAScript contains math objects that can be used for mathematical calculations.

12 Function in item settings

- For a Viedoc item it is possible to write a function in order to assign a particular value to it, either by writing a function that would return a result depending on other items/conditions, or by assigning a default value, as illustrated in the image.
- **Notes!**

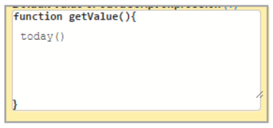
  Functions that returns a value to form item must match the data type as specified in *Data types* section.



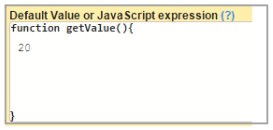
[✓ Function] Default Value

Function logic (JavaScript) (?)

```
function getValue(){

 if( AGE < 18 )
  return 1;
 if( AGE >= 18 && AGE <= 35 )
 return 2;
 if( AGE > 35 )
 return 3;

}
```



```
function getValue(){

 today()

}
```



Default Value or JavaScript expression (?)

```
function getValue(){

 20

}
```

Despite what was earlier mentioned about the matching of data types, nothing will prevent you from writing a non-matching function. Please note that, in case of returning a number with decimals or a date object to a text field, the decimal separator or the date format will be in the format configured on the server, for those cases when the respective function is executed on server side, i.e:

• when applying a new revision.
• for the "auto-update" forms.

13 Debugging your expression

- To debug you can use one of the following in your expression:
    - `debugger;` statement.
    - `console.log('something');`

  The above statements will have effect only when opening the browser’s developer tools while entering data to the respective form in Viedoc Clinic.

  For more information about debugging please refer to http://www.w3schools.com/js/js_debugging.asp.

  Notes!

  • When debugging, you cannot use single line statements.
  • You cannot debug the visit/activity visibility expression as they are run on server.

14 Validation

- During **Save changes** and **VALIDATE** operations in Viedoc Designer, the expressions are validated using a compiler, which would find most of the errors. However, since JavaScript is a dynamic language, not everything can be validated. For example, `AGE.foo ()` will not throw an error, because `AGE` is a variable in a form and the compiler does not know its type.

  Note! The designer must test the expression in all the possible paths using either preview or Viedoc Clinic.