TITOLO DEL DOCUMENTO

December 11, 2025

INDICE

servicenow

1 Introduction to Server-side Scripting

1.1 Categorie di Script

Gli script in ServiceNow si dividono in due categorie principali:

- **Client-side:** eseguiti nel browser dell'utente.
- **Server-side:** eseguiti sul server o sul database di ServiceNow.

1.2 Funzionalità degli Script Server-side

Gli script server-side possono eseguire diverse operazioni, ad esempio:

- Aggiornare campi di record al momento di query sul database
- Impostare valori su record correlati quando un record viene salvato
- Gestire tentativi di login falliti
- Controllare se un utente ha un ruolo specifico
- Inviare email
- Generare e rispondere a eventi
- Confrontare due date per determinare l'ordine cronologico
- Determinare se oggi è un giorno feriale o weekend
- Calcolare la data di inizio del prossimo trimestre
- Scrivere messaggi di log
- Inviare e ricevere messaggi REST da altri sistemi

1.3 Obiettivi del Modulo

Al termine del modulo, imparerai a scrivere, testare e fare debug di due tipi principali di script server-side:

- **Business Rules**
- **Script Includes**

2 Business Rules

2.1 Panoramica

Le **Business Rules** sono script server-side che eseguono logica quando i record di una tabella vengono interrogati, inseriti, aggiornati o cancellati. Queste regole rispondono alle interazioni con il database indipendentemente dal metodo di accesso, come ad esempio:

- Utenti che interagiscono con record tramite form o liste
- Web service
- Importazioni di dati configurabili

Le Business Rules non monitorano i campi dei form, ma vengono eseguite quando un form interagisce con il database, ad esempio al salvataggio di un record.

2.2 Configurazione Principale

- **Name:** Nome della Business Rule.
- **Table:** Tabella del database su cui la Business Rule eseguirà la logica.
- **Application:** Applicazione a cui la Business Rule appartiene.
- **Active:** Abilita o disabilita la Business Rule.
- **Advanced:** Mostra tutte le opzioni di configurazione avanzate.

2.3 Sezione When to Run

- **When:** Definisce quando la logica viene eseguita rispetto all'accesso al database (before, after, instead of).
- **Order:** Ordine di esecuzione tra Business Rules della stessa tabella. Convenzionalmente valori multipli di 100 (100, 200, 300...).
- **Insert:** Esegui quando vengono inseriti nuovi record.
- **Update:** Esegui quando i record vengono modificati.
- **Delete:** Esegui quando i record vengono cancellati.
- **Query:** Esegui quando la tabella viene interrogata.
- **Filter Conditions:** Condizioni da rispettare affinché la logica venga eseguita.
- **Role conditions:** Ruoli richiesti agli utenti che modificano i record.

2.4 Controllare Quando le Business Rules Eseguono

Il campo **When** determina quando, rispetto all'accesso al database, la logica della Business Rule viene eseguita. Le opzioni principali sono:

- **Before:** Esegue la logica prima dell'operazione sul database. Utile per modificare campi del record prima che siano salvati. *Esempio:* Concatenare due campi e scrivere il risultato in un campo Descrizione.
- **After:** Esegue la logica subito dopo l'operazione sul database e prima che il form sia renderizzato. Utile per aggiornare record correlati senza modificare il record principale. *Esempio:* Propagare modifiche dal record padre ai record figli.
- **Async:** Esegue in modo asincrono su un thread separato dopo l'operazione sul database. La transazione corrente continua senza attese. *Esempio:* Invocare web service REST o calcolare SLA.
- **Display:** Esegue la logica quando un form carica un record. Popola l'oggetto `g_scratchpad` per fornire dati al client-side senza modificare il record.

2.4.1 Ordine di esecuzione delle Business Rules

Quando una tabella ha più Business Rules di tipi diversi, l'ordine di esecuzione è tipicamente:

1. Query rules
2. Database query
3. Display
4. Form submit
5. Before rules

6. Database update
7. After rules
8. Async rules

2.5 Azioni delle Business Rules

Le **Business Rule Actions** permettono di configurare azioni eseguite dalla Business Rule senza scrivere script complessi. Le principali azioni sono:

- **Set field values:** Imposta i valori dei campi del record. I valori possono essere:
  - **Dynamically determined (To dynamic):** Valori calcolati a runtime, ad esempio l'utente attualmente loggato.
  - **Same as:** Copia il valore da un altro campo dello stesso record.
  - **Hard coded (To):** Imposta un valore fisso, ad esempio "Awaiting Approval".
- **Add a message:** Visualizza un messaggio in cima al form. Il messaggio può contenere testo formattato, ma elementi multimediali non vengono renderizzati. Utile per notificare l'utente di informazioni importanti o risultati dell'azione.
- **Abort:** Interrompe l'esecuzione della Business Rule e annulla l'operazione sul database. È possibile mostrare un messaggio usando l'opzione "Add message" per informare l'utente.

2.5.1 Esempio pratico

- "Requested for" = valore dinamico (utente corrente)
- "Description" = valore uguale a "Short description"
- "State" = valore hard coded "Awaiting Approval"
- Aggiungere un messaggio informativo sul form
- In caso di condizione non soddisfatta, usare "Abort" + messaggio

3 Business Rule Scripts

3.1 Introduzione

I **Business Rule Scripts** consentono di eseguire logica server-side quando un record viene interrogato, inserito, aggiornato o cancellato. Per scrivere script è necessario selezionare l'opzione **Advanced** nella configurazione della Business Rule.

3.2 Campi di scripting

- **Condition:** Espressione booleana che determina se il codice dello script deve essere eseguito.
- **Script:** Codice server-side vero e proprio, eseguito solo se la condizione è vera.

3.3 Oggetti principali

- **current:** Record corrente con tutti i campi e metodi GlideRecord.
- **previous:** Valori del record prima della modifica (non disponibile per async Business Rules).

3.4 Esempi di condizioni

- Verifica se il campo `short_description` ha un valore specifico:

  ```
  current.short_description == "Hello world"
  ```

- Verifica se il campo `state` è cambiato al valore 6:

  ```
  current.state.changesTo(6)
  ```

- Verifica se il campo `short_description` non è vuoto:

  ```
  !current.short_description.nil()
  ```

- Verifica se il campo `short_description` è cambiato rispetto al precedente:

  ```
  current.short_description != previous.short_description
  ```

3.5 Esempio di script

```
(function executeRule(current, previous /*null when async*/) {
  // Se la descrizione non è cambiata, non fare nulla
  if (current.description == previous.description) {
    return;
  }

  // Aggiunge un suffisso alla descrizione
  current.short_description = current.short_description + " - verificato";

  // Imposta un valore hardcoded su un altro campo
  current.state = 6;

})(current, previous);
```

4 Dot-Walking

4.1 Introduzione

Il **Dot-Walking** permette di accedere ai campi dei record correlati direttamente tramite scripting. I campi di tipo *Reference* contengono lo `sys_id` di un record in un'altra tabella, e dot-walking consente di leggere o scrivere valori senza query aggiuntive.

4.2 Sintassi

```
<oggetto_corrente>.<campo_reference>.<campo_del_record_correlato>
```

4.3 Esempi

- Controllare l'email di un utente referenziato:

```
if(current.u_requested_for.email == "beth.anglin@example.com"){
  // logica qui
}
```

- Accedere a campi più profondi:

```
var latitudine = current.u_requested_for.company.latitude;
```

4.4 Script Tree

Il **Script Tree** permette di navigare tra i campi dei record correlati. Cliccando su un campo, ServiceNow inserisce automaticamente la sintassi di dot-walking nello script.

5 Server-side APIs

Le **Server-side APIs** sono classi e metodi forniti da ServiceNow per interagire con il database e con la piattaforma direttamente dal server. Queste API vengono utilizzate nei *Business Rules*, *Script Includes*, *Inbound Email Actions* e altri script lato server.

6 Current, Previous e GlideSystem in Business Rules

6.1 Oggetti disponibili

- **current**: rappresenta il record attuale su cui viene eseguito lo script. Contiene tutti i campi aggiornati in tempo reale.
- **previous**: rappresenta lo stato del record prima della modifica. Utile per confrontare i valori vecchi e nuovi. Non disponibile nelle Business Rules `async`.
- **gs (GlideSystem)**: oggetto di sistema per funzioni generali come logging, messaggi all'utente, generazione eventi, ecc. Non gestisce direttamente `current` o `previous`.

6.2 Esempio di utilizzo

```
// Se lo stato del record è cambiato a 6
if(current.state.changesTo(6)) {
    gs.addInfoMessage("Lo stato del record è diventato 6!");
}

// Confronto current e previous
if(current.short_description != previous.short_description){
    gs.log("La descrizione è cambiata!");
}
```

6.3 Note

- `current` e `previous` sono oggetti `GlideRecord`.
- `gs` è indipendente dai record e fornisce metodi generici di sistema.

6.4 GlideSystem (gs)

Permette di accedere a informazioni generali sul contesto di esecuzione.

- `gs.getUserName()` - restituisce il nome dell'utente corrente.
- `gs.nowDateTime()` - restituisce la data e ora corrente.
- `gs.addInfoMessage("Messaggio")` - aggiunge un messaggio informativo alla UI.
- `gs.log("Messaggio")` - scrive un messaggio di log sul sistema.

6.5 GlideRecord

Serve per leggere, creare, aggiornare o cancellare record nel database.

```
// Esempio di utilizzo di GlideRecord
var gr = new GlideRecord('incident');
gr.addQuery('state', 1); // aggiunge un filtro
gr.query();
while(gr.next()){
    gs.log(gr.number);
}
```

- Metodi principali: `get()`, `update()`, `insert()`, `deleteRecord()`.
- Supporta il **dot-walking** per accedere ai record correlati.

6.6 GlideDateTime

Gestisce data e ora in ServiceNow.

- Creazione oggetto data/ora: `var gdt = new GlideDateTime();`
- Aggiunta giorni: `gdt.addDays(3);`
- Formato leggibile: `gdt.getDisplayValue();`

6.7 Note importanti

- Le API sono **release-specific**: alcune funzioni possono variare tra versioni (es. San Diego vs Utah).
- Esistono versioni **scoped** e **global** delle API: usare quella corretta in base al contesto dell'applicazione.

7 GlideSystem API

La **GlideSystem** (gs) è un oggetto server-side fondamentale in ServiceNow che permette di:

- Accedere a informazioni sull'utente corrente e sul contesto.
- Loggare messaggi di diversi livelli (debug, info, warning, error).
- Mostrare messaggi all'utente nella UI.
- Generare eventi.
- Eseguire controlli temporali e calcoli sulle date.

7.1 Esempio di utilizzo

Il seguente script mostra come usare alcune funzionalità di GlideSystem:

```
// Scrive un messaggio nel log di sistema
gs.log("Messaggio di log");

// Mostra un messaggio informativo nella pagina dell'utente
gs.addInfoMessage("Questo è un messaggio informativo");

// Mostra un messaggio di errore nella pagina dell'utente
gs.addErrorMessage("Questo è un messaggio di errore");
```

7.2 Note

- `gs.log()` invia i messaggi al log di sistema visibile in *System Logs* > *All*.
- `gs.addInfoMessage()` e `gs.addErrorMessage()` scrivono messaggi visibili all'utente nella pagina corrente.
- Altri metodi utili includono `gs.eventQueue()` per generare eventi e `gs.nowDateTime()` per ottenere la data/ora corrente.

8 GlideRecord

8.1 Creazione e query

```
// Creazione oggetto GlideRecord
var gr = new GlideRecord('table_name');

// Aggiunta condizioni
gr.addQuery('state', 14);
gr.addQuery('active', true);

// Esecuzione query
gr.query();
```

8.2 Iterazione sui record

```
// Itera tutti i record
while(gr.next()){
```

```
    gs.info(gr.number + " - " + gr.short_description);
}

// Aggiornamento record singolo
if(gr.next()){
    gr.priority = 4;
    gr.update();
}

// Aggiornamento multiplo
gr.setValue('priority', 4);
gr.updateMultiple();
```

8.3 Query complesse e encodedQuery

```
var gr = new GlideRecord('x_snc_needit_needit');
gr.addEncodedQuery(
  "u_when_neededBETWEENjavascript:gs.daysAgoStart(0)@javascript:gs.quartersAgoEnd(1)^active=true^state=14^
);
gr.query();
while(gr.next()){
  // Logica sui record
}
```

8.4 Note

- 'addQuery' senza operatore assume '='.
- 'getRowCount()' può avere impatto sulle performance, usare 'GlideAggregate' in produzione.
- Le condizioni multiple in 'addQuery' sono AND di default; usare encodedQuery per OR complessi.

9 GlideDateTime

9.1 Creazione oggetti

```
// Ora corrente
var gdt = new GlideDateTime();

// Data/ora specifica
var gdt2 = new GlideDateTime('2025-12-10 15:00:00');
```

9.2 Confronto date

```
if (gdt.after(gdt2)) {
    gs.info('gdt è più recente di gdt2');
}
```

9.3 Operazioni su date

```
// Aggiunge 3 giorni
gdt.addDaysLocalTime(3);

// Aggiunge 60 secondi in UTC
gdt.addSecondsUTC(60);
```

9.4 Formattazione e conversione

```
var localString = gdt.getLocalDate();
var gmtString   = gdt.getDisplayValue();
```

9.5 Note importanti

- Alcuni metodi usano GMT/UTC, altri l'orario locale.
- Alcuni metodi ritornano millisecondi.
- Per calcoli semplici su giorni usare anche `gs.daysAgo()`.

10 Debugging Business Rules

10.1 Introduzione

Il debugging delle Business Rules permette di verificare il comportamento dei script lato server quando i record vengono inseriti, aggiornati, eliminati o letti.

10.2 Metodi principali di debug

- **Script Editor:** Controlla errori di sintassi JavaScript e problemi banali.
- **System Logs:** Usa i metodi della classe `GlideSystem` (`gs`) per loggare informazioni.
  - `gs.info("Messaggio informativo");`
  - `gs.warn("Messaggio di warning");`
  - `gs.error("Messaggio di errore");`
  - `gs.debug("Messaggio di debug"); # deve essere abilitato tramite proprietà dell'app`

  I log appaiono in **System Logs** > **Application Logs**.
- **Debug Business Rule (Details):** Mostra quando una Business Rule viene eseguita o saltata, inclusi i motivi per cui la `Condition` non è soddisfatta. I log sono visibili anche nella form o lista dove la Business Rule viene eseguita.
- **JavaScript Debugger:** Permette di usare breakpoints, eseguire step into/out/over, e monitorare variabili durante l'esecuzione del codice lato server.

10.3 Tip pratici

- Per attivare i messaggi di debug, creare o modificare la proprietà dell'app:
  - **Nome:** `<app_scope>.logging.verbosity`

  - **Valore:** `debug` (oppure lista separata da virgola: `info,warn,error`)
- Usare la funzione appropriata di `gs` in base alla gravità del messaggio.
- Abilitare il debug dettagliato per analizzare la valutazione delle condizioni.

10.4 Esempio di Business Rule con logging

```
// Log di diversi livelli
gs.info("Informazione: Short description = " + current.short_description);
gs.warn("Attenzione: il record ha stato non previsto");
gs.error("Errore: campo obbligatorio mancante");

// Debug (assicurarsi che <app_scope>.logging.verbosity = debug)
gs.debug("Debug: valori correnti dei campi");
```

11 Script Tracer

11.1 Introduzione

Lo **Script Tracer** è uno strumento del *Script Debugger* che permette di tracciare l'esecuzione dei script server-side sincroni, inclusi quelli legati a UI Actions o transazioni lato applicazione.

11.2 Componenti principali

- **Search field:** per cercare script o transazioni specifiche.
- **Filters:** per filtrare tipi di script o tabelle.
- **Start/Stop button:** avvia e ferma il tracciamento.
- **Trace status:** mostra se il tracing è attivo.
- **List of changes:** elenca tutti gli script e le UI Actions eseguite.
- **Details:** visualizza errori o modifiche dei campi.

11.3 Come utilizzare

1. Aprire *System Diagnostics* > *Script Tracer* dal menu **All**.
2. Cliccare **Start Tracer**.
3. Tornare al form o alla lista da testare e eseguire l'azione che attiva lo script (ad esempio, clic su **Save** o **Update**).
4. Lo Script Tracer traccia l'esecuzione delle transazioni server-side.

11.4 Visualizzazione dei risultati

- **State tab:** mostra i campi e i valori modificati, evidenziando in verde i cambiamenti.
- **Script tab:** mostra le linee di script associate all'esecuzione; con **Show Script** si visualizza l'intero script.
- **Transactions tab:** dettaglia la transazione corrente, inclusi i record modificati e le operazioni eseguite.
- **Errori:** indicati con un cerchio rosso contenente una X.

11.5 Tip pratici

- Cliccare **Debug Script** per aprire lo script selezionato nel Script Debugger.
- Usare **View File** per modificare il record dell'UI Action direttamente nella piattaforma.
- Deselect "Show only changed values" per vedere tutti i campi del record e non solo quelli modificati.

12 Script Debugger

12.1 Introduzione

Lo **Script Debugger** è lo strumento principale per il debug dei Business Rules e altri script server-side sincroni. Permette di esaminare l'esecuzione dello script in dettaglio e di controllare il flusso di esecuzione.

12.2 Funzionalità principali

- Impostare, rimuovere e mettere in pausa i **breakpoints**.
- Eseguire il codice **linea per linea**.
- Entrare (**step into**) ed uscire (**step out**) da funzioni e chiamate a metodi.
- Visualizzare valori di variabili **locali, globali e private**.
- Visualizzare lo **stack di chiamate** (*call stack*).

12.3 Componenti principali del debugger

- Pannello del codice con evidenziazione della linea corrente.
- Pannello delle variabili che mostra i valori aggiornati in tempo reale.
- Call stack per analizzare la sequenza di chiamate.
- Controlli di esecuzione: Step Over, Step Into, Step Out, Resume, Pause.

12.4 Accesso e requisiti

- Solo utenti con i ruoli **admin** o **script_debugger** possono utilizzare lo Script Debugger.
- Modalità di apertura:
  - **All menu** → System Diagnostics → Script Debugger
  - In **Studio**: File → Script Debugger
  - Dal **Script editor toolbar**: cliccare il bottone *JavaScript Debugger*
  - Dalla finestra principale: avatar utente → Preferences → Debugging card → Open Script Debugger

12.5 Tip pratici

- Usare breakpoints per fermare l'esecuzione in punti critici e analizzare valori.
- Combinare lo Script Debugger con lo *Script Tracer* per vedere quali script vengono eseguiti in una transazione.
- Monitorare le variabili *current* e *previous* nei Business Rules per capire come cambiano i valori dei record.

13 Breakpoints

13.1 Introduzione

I **breakpoints** fermano l'esecuzione dello script per permettere agli sviluppatori di esaminare variabili e valori in quel punto. Lo **Script Debugger** mostra nella barra di stato se uno script è in pausa a causa di un breakpoint o in attesa di raggiungerne uno.

13.2 Tipi di Breakpoints

- **Breakpoints normali**: si fermano sempre alla linea specificata.
- **Breakpoints condizionali**: si fermano solo quando lo *script condizione* restituisce true.

13.3 Impostazione dei Breakpoints

- Breakpoints session-specific: influenzano solo la sessione dello sviluppatore che li ha impostati.
- Impostazione:
  - Cliccare nel **gutter** (colonna a sinistra) della linea di interesse.
  - Oppure clic destro → Add breakpoint.
- Rimozione: clic o clic destro sul breakpoint.
- Condizioni: clic destro sul breakpoint → Add conditional breakpoint → inserire condizione in JavaScript server-side (es. `current.short_description != ""`).

13.4 Indicazioni visive

- **Breakpoint normale**: freccia viola + sfondo grigio.
- **Breakpoint condizionale**: freccia gialla + sfondo grigio.
- Hover sulla freccia gialla mostra la condizione dello script.

13.5 Controlli durante il debug

Quando lo script è in pausa a un breakpoint:

- **Pause**: ferma qualsiasi sessione di debug e disabilita lo Script Debugger.

- **Start debugging**: abilita lo Script Debugger e pausa sui breakpoints.
- **Resume**: continua fino al prossimo breakpoint o alla fine dello script.
- **Step Over**: esegue la funzione senza fermarsi sulle linee interne.
- **Step Into**: entra nella funzione/method call e ferma sulla prima linea eseguita.
- **Step Out**: esce dalla funzione corrente e ritorna al chiamante.

13.6 Call Stack

Mostra l'elenco dei metodi e delle funzioni chiamate durante l'esecuzione. Click su un elemento per visualizzare la definizione. Deve esserci uno *pause* attivo per esaminare lo stack.

13.7 Transaction Details

Quando lo script è in pausa, i **Transaction Details** mostrano:

- URL della richiesta
- Session ID
- Utente
- Parametri della richiesta
- Istanza
- Orario di inizio e altri dettagli della transazione

14 Session Log

14.1 Introduzione

Il **Session Log** in ServiceNow è uno strumento di debug per visualizzare informazioni dettagliate sui server-side scripts eseguiti durante una sessione. È utile per tracciare:

- Business Rules
- SQL
- Escalations
- Data Policies
- Date/Time
- Log, Security, Quotas
- GraphQL e Security Rules
- Homepage Render

14.2 Componenti principali

- **Transaction list**: elenco delle transazioni registrate.
- **Filters**: permette di filtrare i log per tipo, applicazione o transazione.
- **Settings**: configura quali eventi tracciare (es. Business Rules, Data Policies, SQL, ecc.).
- **Log text area**: area dove appaiono i messaggi registrati.

14.3 Funzionamento

1. Aprire il **Session Log** nel *Script Debugger*.
2. Cliccare **Settings** per selezionare i tipi di eventi da tracciare.
3. I log appaiono nell'area di testo.
4. Filtrare i log per visualizzare solo i messaggi rilevanti (Info, Warn, Error, Debug).

14.4 Suggerimenti d'uso

- Configurare i Settings prima di avviare il debug.
- Usare i filtri per concentrarsi sui messaggi più utili.
- Combinare con Breakpoints e Script Tracer per avere una panoramica completa.