Stored Procedures & Triggers

####Not yet implemented/released####

Introduzione

Le *Stored Procedures*, così come i *Trigger*, non fanno parte della natura del server eXtraWay ma ne rappresentano una forma dinamica d'estensione.

Da molto tempo a questa parte esiste la possibilità di realizzare una “DLL d'archivio”, vale a dire una libreria dinamica che viene chiamata dal server, se presente e configurata, per svolgere azioni specializzate in alcuni punti ben definiti[1] per i quali sono stati realizzati degli *entry point* detti morsetti.

Col passare del tempo è risultato evidente che una simile realizzazione scarica su 3D Informatica tutto l'onere dello sviluppo e della manutenzione anche per azioni tutto sommato molto semplici, da svolgersi sia per completare/perfezionare i record che vengono salvati sia per *decorare* quelli che vengono mostrati.

Per questa ragione, sempre con la logica della libreria dinamica da evocare al momento giusto nel modo giusto, è stata realizzata una coppia di librerie dinamiche, libxwlua.[dll|so][2] e libxwluafunc.[dll|so][3], che assolve a questi compiti.

Una rapida Overview

E' disponibile una rapida *Overview* in questo file PDF.

Componenti dell'architettura software

Le due librerie devono essere collocate come segue:

File	Collocazione
liblua.[dll|so]	Directory `plug-in` presente dentro alla directory degli eseguibili eXtraWay, ovvero nella directory ove si trova il server.
libluafunc.[dll|so]	La collocazione di questa libreria dinamica varia da piattaforma a piattaforma. Dal momento che in Windows viene sempre annoverato tra le directory ove tentare il reperimento di librerie dinamiche anche quella dalla quale viene lanciato l'eseguibile principale, si suggerisce di collocare il modulo libluafunc.so nella directory dei binari di eXtraWay, quindi la stessa in cui si trova il server. Per quanto concerne Linux bisogna assicurarsi che la libreria dinamica sia un un *Path* riconosciuto, ragion per cui si rimanda al file di configurazione `xwclt.conf`, di cui fa uso lo script `xwctl` che viene normalmente usato per eseguire i servizi eXtraWay su piattaforma Linux. In tale file si cita con una voce di configurazione, la directory ove identificare le librerie mancanti nei percorsi standard.

Versioni (Lua Batteries)

2.0.X Candidate

- **2.0.24**
  - Introdotto correttamente l'uso della directory di *share* in sede di esportazione. Richiede Server 25.9.2-SE ovvero 1.0.3-EE.
- **2.0.23**
  - Test. Introdotto test per MySQL e SQLite3. Separati test standard da test “eXtraWay depending” per poterli eseguire indipendentemente.
- **2.0.22**
  - Corretto il comportamento della `csv.lua` in modo che accetti file con riporti a capo in stile *Windows*[4] anche su macchine linux senza accodare un carattere indesiderato all'ultima colonna.
- **2.0.20/21**
  - Modfiche per il passaggio a 64 bit e l'uso della documentazione del nuovo LDT.
- **2.0.19**
  - Introdotto un nuovo metodo per la determinazione della cartella dei file *Shared* del server, ovvero al cartella, distinta da quella dei file temporanei, ove il server produce i file condivisi con il mondo esterno, come ad esempio i file di esportazione.
- **2.0.18**
  - Introdotto un nuovo parametro (opzionale) ai processi di esportazione CSV in modo da richeidere che venga preposto il `BOM UTF-8` in testa al risultato ottenuto così da poterlo esportare con indicazione dell'encoding del contenuto.
- **2.0.17**
  - Introdotto metodo per la scelta del log sul quale compiere le registrazioni.

2.0.16

Emessa il 02/07/2014

- **2.0.13**

- Nella stesura dei file CSV non si teneva conto correttamente del fatto che un testo potesse contenere dei riporti a capo.

- **2.0.14**
  - Nella creazione dei file Zip non venivano impostati correttamente i diritti per un utilizzo nell'ambiente Linux. Così facendo, pur producendo uno .zip apparentemente valido e potendo estrarre da esso i file contenuti, tali file risultavano inaccessibili.
  - L'esportazione non riconosce più `item` e `group` bensì `item` e `collection`[5)].
- **2.0.15**
  - Reintrodotto il metodo `xw.callPlugIn()` che era stato rimosso durante il *refactoring*.
- **2.0.16**
  - Creato attorno al processo di esportazione un *Wrapper* che consenta di farne uso sia come chiamata diretta ad una *Stored Procedure* che produce l'esportazione sia da un'altra *Stored Procedure* o da un *Trigger* quindi in modalità interna.

Nel mese di giugno è stata elaborata una soluzione espressamente per Centos in quanto la libssl.so di tale distribuzione è volutamente incompleta (viene realizzata escludendo alcuni algoritmi per ragioni di licenza) e non consente l'uso regolare delle mail via LUA.
Per ovviare all'inconveniente sono state compilate da sorgenti standard due librerie (libssl.so.1.0.0 e libcrypto.so.1.0.0) da collocare nella cartella `libs` in modo che il nostro sistema le utilizzi in sostituzione a quelle originali.
Da applicarsi esclusivamente su distribuzioni Centos.

2.0.12

Emesso il 22/05/2014

- Nuovo corso degli script lua, compatibile con versione della libreria 2.0.0 o superiori.
  Comprende molteplici componenti tra cui un *Mail Sender*.
- Introdotto il supporto all'interpretazione di righe provenienti da una fonte CSV e per la produzione di righe analoghe.
- Creato un processo di esportazione generico (ed uno di importazione specializzato) da e per CSV.
- Introdotta la gestione dei file Zip (scrittura).
- Il sistema di invio mail può essere anche disabilitato.

[Patch 1] Modificato il file 'relational.lua' in modo che gestisca elementi `collection` e non `group`.

Versioni (Moduli Binari)

2.1.1 Candidate

Lua Batteries 2.0.19 e superiori

- Introdotto metodo per la determinazione della cartella dei file *Shared* da usare, ad esempio, per le esportazioni. Tale cartella, se configurata, differisce dalla cartella dei termpoanei ed è solitamente condivisa con il mondo esterno.
- Consentito di accedere a metodi di una tabella locale tornata dallo script lua e non solo i metodi della tabella globale con lo stesso nome del package.
- Corretto errore nella node::selectNodes() che causava un *hang* del server se l'espressione `XPath` non era corretta.
- Inibito il trasferimento di parametri in modalità POST nei comandi HTTP quando essi non sono stati forniti (o risultano vuoti) per evitare di confliggere con servizi che non si supportano.

2.1.0 Candidate

Lua Batteries 2.0.17 e superiori

- Introdotto metodo per scegliere il log ove compiere le registrazioni
- Corretto il trattamento dei nodi in sede di replica in quanto causava un *Memory Leak*.
- Corretto un problema di uscita dallo `scope` di un oggetto di tipo *record* mentre i suoi `nodi` erano ancora in uso.

2.1.0 Candidate

Lua Batteries 2.0.0 e superiori

- La libreria dev'essere allineata al server: si richiede versione 25.7.0 o superiore.
- Al fine di evitare il rischio di conflitto nella denominazione dei moduli ora la `liblua.[dll|so]` e la `libluafunc.[dll|so]` si chiamano rispettivamente `libxwlua` e `libxwluafunc`.

2.0.1

Lua Batteries 2.0.0 e superiori

- Corretto il trattamento dei nodi in sede di replica in quanto causava un *Memory Leak*.
- Corretto un problema di uscita dallo `scope` di un oggetto di tipo *record* mentre i suoi `nodi` erano ancora in uso.

2.0.0 (Era 1.13.0)

Emesso il 22/05/2014
Lua Batteries 2.0.0 e superiori

- Forzato il caricamento della `libluafunc.[dll|so]` in modo che si acquisisca indipendentemente dalle configurazioni dei *path* della macchina.
  Questo ha lo scopo di garantire la convivenza di più installazioni sullo stesso server anche se dovessero far uso di versioni diverse senza rischiare di condividere tale libreria.
  Per effetto di questa modifica ora il file `libluafunc.[dll|so]` **deve** risiedere nella cartella dei binari di eXtraWay e non, come avveniva per le insallazioni **Linux** fino ad ora, un una qualche cartella raggiungibile o indicata dalla variabile `xw_ldpath` nel file `xwxctl.conf`, solitamente riferita ad una cartella `libs` parallela a quella dei binari di eXtraWay.
  Si abbia quindi cura, in sede di installazione o di aggiornamento, di ignorare (rimuovere) ogni eventuale libreria `libluafunc.so` presente della cartella `libs`.
- Introdotti nuovi *Triggers* che precedono e seguono la composizione dei titoli..
  Il primo viene evocato con la regola di composizione del titolo così che la si possa modificare secondo le proprie esigenze.
  Il secondo riceve il numero del documento corrente, la stringa del titolo calcolata e la regola di composizione del titolo utilizzata.
- La cancellazione dei nodi di un nodeSet può sfociare in errore se i nodi sono nidificati tra loro.
  Implementata una modalità di cancellazione massiva dei nodi che dovrebbe tutelare la struttura del record ed evitare problemi. Richiede versione del server 25.5.0 o superiore.
- Modificato il comportamento delle funzioni `setValue()` e `setAttribute()` degli `xwLuaNode_t` in modo che causino un errore bloccante se si cerca di assegnare un valore in un *encoding* riconosciuto come chiaramente errato al fine di tutelare l'intero ambiente dal rischio di errori silenziosi e difficili da rilevare.
- Introdotti metodi per l'accesso ai vocabolari e la loro navigazione ed analisi spettrale.
- Introdotto un metodo per acquisire i dettagli (meta dati) di un record, quali data ed ora di inserimento e modifica, data ed ora e nome del file XML ove il record risiede e simili.
- Modificati i metodi per mezzo dei quali si accede ai titoli perché tornino, oltre al titolo in se, anche il numero fisico del documento di un `xwLuaRecordSet`.
- Introdotta la possibilità di compiere lock di record che sopravvivono all'esecuzione dello script[6] per mantenere un record bloccato sino a successive operazioni su di esso. Richiede server eXtraWay versione 25.5.2 o superiore.
- ***Windows Only*: Passaggio dalla versione 2.0.0 alla 2.0.2 di Lua. Richiede aggiornamento moduli `liblua.dll`, `libluafunc.dll` e `lua51.dll`**.
- **Passaggio ad una nuova architettura dell'uso di Lua che rende incompatibile col passato molti moduli e richiede interventi sulla `liblua.dll`, `libluafunc.dll` e `lua.dll`(lua51.dll viene abolita) e la corrispondente versione Linux[7].**
  Server 25.6.0 o superiore..

1.12.0

Emesso il 14/01/2014

- Corretto errore di *Stack Overflow* in operazioni cicliche.
- Introdotto metodo per forzare la *Garbage Collection* dal momento che nei processi ciclici nei quali si compia un elevato numero di operazioni LUA stesso non fa in tempo a mantenere la memoria libera. Richiede allineamento al server 25.3.0 o superiore.
- Introdotto nuovo set di comandi per realizzare un record partendo da un file o da uno string buffer[8].
- Introdotto comando 'applyXsl' nell'oggeto xwLuaRecord_t in modo da poter applicare fogli di stile `xsl` su record esistenti quando questo processo (seppure più lento) risulti vantaggioso per la mole di attività da compiere.
- Introdotto comando 'calcDigest' tra i comandi generici del server in modo da poter calcolare l'impronta di una serie di file ed in questo modo implementare un servizio che si occupi di questo per qualsivoglia soluzione documentale.
- Introdotto flag di ordinamento nel comando 'getChildren()' dell'oggetto `xwLuaNode_t`.
- Modificato il metodo di `lock()` in modo che torni un valore `#boolean` che indichi se il lock è andato a buon fine o meno.
- Introdotta la possibilità di evocare, in uno script LUA, una funzione di una DLL d'archivio o genericamente del server. Richiede allineamento al server 25.3.0 o superiore.
- Reso statico l'utilizzo della libreria `libcurl` entro la `libluafunc` così da superare ogni problema di caricamento.

1.10.0

Emesso il 09/10/2013

- Il *renaming* di un elemento non produce un elemento correttamente rinominato e valido. Corretto.
- Abolito il metodo `getRecord` del tipo `xwLuaNode`..
- Modificati i metodi `lock` e `reload` del tipo `xwLuaRecord`. Ora l'operazione non torna più un nuovo record ma mantiene aggiornato (con la condizione di lock o con il contenuto ricaricato) direttamente nel record stesso.
- Introdotto metodo di *Garbage Collection*.

1.8.0

Emesso il 15/05/2013

- Revisione completa del sistema di gestione dei *Triggers* in modo da ammettere la possibilità di spegnere ed accendere i *Triggers* utente separatamente dai *Triggers Amministrativi* sui quali si può agire esclusivamente quando si è già in ambito autoritativo (per evitare *infinite loop*).
- Rettifica del sistema di logging perché si comporti correttamente per logs su porta non standard.
- Introdotto metodo per la determinazione del prossimo valore seriale (xw.getNewSerial()).

1.6.2

- Introdotto metodo per assegnare un file ad un archivio come nuovo allegato. Il metodo torna il codice di allegato da utilizzare in un record (elemento `xw:file`, attributo `name`) per perfezionare l'accoppiamento tra allegato e record.

1.6.1

Emesso il 11/04/2013

- Corretto errore di esecuzione rientrante quando un *Trigger* o una *Stored Procedure* causano indirettamente l'esecuzione ex-novo dello stesso metodo.

1.6.0

Emesso il 08/04/2013

- Introdotto un comando `ping` nel *package* di default `xwTools` così da verificare il comportamento del sistema.
  Scrive nel log e torna un messaggio di corretto funzionamento.
- Riveduto il processo di spegnimento dei *Trigger* in modo che tenga conto di operazioni di `Off/On` nidificate.
- Introdotto *Garbage Collection* per i principali oggetti *eXtraWay* generati ed usati negli script *Lua*.
- Introdotti metodi per chiamate http.
- Introdotta la possibilità di sfruttare *Stored Procedure* per files sottoposti al server via *Watch Doc*[9].
- Introdotti comandi per l'identificazione del Sistema Operativo.

1.4.0

Emesso il 07/02/2013

- Modificata la modalità per mezzo della quale si calcola il nome dei files temporanei al fine di evitare pericolose omonimie.
- Modificato il valore tornato in caso di valore nullo per fornire un valore che sia sempre XML valido e quindi non impatti sulla qualità del risultato.
- Introdotto nuovo *Trigger* che si può utilizzare alternativamente all'estensione `xslt` del documento. Si attiva introducendo la parola `lua` al posto del nome del file `xslt` da utilizzare nell'impostazione `ud_xslt` del file di profilo dell'archivio. Viene evocato in sede di costituzione del catalogo, inserimento o modifica di un documento e deve comportarsi come si comporta l'estensione `xslt` vale a dire deve comporre una porzione supplementare di record da indicizzare, usare nei titoli e/o negli ordinamenti.
- Introdotto metodo per la determinazione della directory dei files temporanei utilizzata da eXtraWay.
- Introdotto un nuovo *Trigger* che viene evocato prima della costituzione di un titolo. Ci si riferisce alla costituzione reale, fisica, di un titolo, quella che viene posta nella cache. Si possono in questo modo realizzare estensioni del documento, creandone parti che in realtà non esistono, e che hanno finalità esclusivamente nell'ambito del titolo. Può essere sfruttato per l'ordinamento ma non per gli indici (per i quali si prevede un diverso *Trigger*).
  Risponde all'evento `onTitle` ovvero `<primary_node>.onTitle`.
- Analogmente al precedente `onTitle` è stato implementato un *Trigger* `onSort` che opera sul documento quando questo viene caricato per speciali attività di ordinamento che non sarebbero possibili senza una rielaborazione dei contenuti. Come il precedente, questo *Trigger* non può e non deve interagire con altri documenti.

1.2.0

Emesso il 14/12/2012

- Modificata la modalità di accesso ai documenti così da poter impostare vari flags di caricamento al bisogno.
- Oltre alle funzioni di copia di nodi in un DOM ora è possibile anche compiere operazioni di spostamento degli stessi.
- Corretto il comportamento della record:setContentFromString() che non convalida un record al quale si assegna un valore se non già precedentemente valido.
- Corretta la funzione di caricamento di un documento singolo perché faccia correttamente hilight e call dll.

1.1.5 Candidate

- Introdotta la possibilità di scartare un record con la chiamata `record:discard()` liberando le risorse ad esso relative.
  Non potendo compiere una effettiva *Garbage Collection* ci si può per lo meno tutelare dall'uso incontrollato di Ram.

- In caso di ritorno multiplo di parametri da parte di una *Stored Procedure* essi vengono acquisiti nell'ordine inverso rispetto a quello previsto e dichiarato nel package stesso. Ripristinato il corretto ordine di valutazione.

1.1.2/3/4 Candidate

- Introdotta la possibilità di caricare un record con *high-light* dei termini oggetto della selezione.

1.1.1 Candidate

- Introdotto il metodo per ottenere l'ID di un *Result Set*.
- Introdotti metodi `get[First|Next|Prev|Last|Current|Absolute]Title()` e `setTitleRule` per ottenere, da un `xwLuaResultSet` i titoli dei documenti secondo il criterio preferito.
- Introdotto il tipo xwLuaMap che consente di costituire mappe associative di parametri per chiamate complesse al server. Attualmente sono presenti i metodi di alimentazione ma non di consultazione.
- Introdotta la chiamata “indiretta” ad una *Stored Procedure* appartenente presumibilmente ad un diverso archivio (ma anche all'archivio corrente).
  L'effetto ottenuto è che questa chiamata indiretta opera entro un *Runner* distinto da quello corrente e, se si tratta di *Stored Procedure* di archivio diverso, essa non viene contaminata con l'*environment* corrente, in particolare con l'archivio corrente.

1.1.0 Candidate

- Revisione ed arricchimento della documentazione.
- La condizione d'errore in esecuzione di un Trigger, specie in caso di trigger d'autorizzazione, non veniva correttamente intercettata e quindi il processo proseguiva anche non dovendo.
- La chiamata di un trigger in modalità *fallback*[10] non veniva correttamente effettuato seppure l'identificazione del metodo fosse corretta.

1.0.0

Emesso il 14/11/2012

- Prima realizzazione delle funzionalità Lua supportate;
- Introdotta la gestione dei diritti che consente di conoscere le *label* dei diritti assegnati ad un utente o, se omesso, all'utente corrente;
- Ampia revisione del comportamento, della configurazione e della documentazione. Approssimazione al rilascio della versione 1.0.0.
- Link diretto con script Lua di sistema e verifica della versione.

Linguaggio di Scripting

Per introdurre *Stored Procedures* e *Triggers* tra le funzionalità del Server eXtraWay abbiamo ritenuto di doverci avvalere di un linguaggio di *scripting* potente e di ampia diffusione.
Per tale ragione la scelta è caduta su Lua.
Si assume quindi che il lettore che voglia intraprendere la stesura di proprie *Stored Procedures* o *Triggers* acquisisca una ragionevole padronanza del linguaggio avvalendosi del suo Tutorial o del Reference.

Può risultare inoltre vantaggioso prendere visione di alcune implementazioni già esistenti quali:

- Alcuni esempi di codice già sviuppato
- Alcuni comandi famosi "tradotti" in vari linguaggi, su wikipedia
- Interprete online di Lua, per esperimenti

NOTA BENE
In seguito verranno mostrati i diversi metodi disponibili per i tipi evoluti di dato che sono Record, Nodi del DOM e rispettivi Set.
Gli sviluppatori che sono soliti avvalersi di linguaggi evoluti, ad oggetti, sanno bene che evocare un metodo di un oggetto comporta che l'oggetto stesso viene eventualmente modificato, cambia stato, muta in senso generico.
In Lua, seppure le chiamate dei metodi fanno pensare che una chiamata su un tipo evoluto produca un effetto sull'istanza stessa, non è così.
Abbiamo quindi metodi che ci impongono, una volta evocati, di raccogliere un valore di ritorno che rappresenta la vera mutazione.
Alcuni esempi sono i comandi `record:lock()` ovvero `record:setContentFromString` o `record:setContentFromRecord`.

Librerie di funzionalità: i Packages

La differenza sostanziale tra le *Stored Procedure* ed i *Trigger* è presto detta:
Mentre le prime vengono evocate esplicitamente nel momento più opportuno e sotto un controllo diretto da parte dell'applicazione che ne fa richiesta, i secondi vengono evocati direttamente dal server ed indipendentemente dalla “volontà” dell'operatore.
I *Trigger* si distingueranno quindi in *Trigger d'utente* e *Trigger d'autorizzazione* come vedremo meglio in seguito.
Oltre a questo, le *Stored Procedure* prevedono parametri in ingresso ed in uscita che sono del tutto liberi mentre per i *Trigger* essi devono rispettare uno standard discusso in seguito.

Quale che sia la natura degli obiettivi che ci si prefigge, *Stored Procedure* e *Trigger* sono metodi scritti in linguaggio Lua che devono essere sfruttabili dal server *eXtraWay*.
Perché questo avvenga è indispensabile che il server possa rispondere correttamente alle seguenti domande:

- Come si identifica in modo univoco lo script Lua che deve contenere il metodo che intendo utilizzare?
- Come si identifica in modo univoco l'elenco dei parametri di input/output di ciascun metodo?
- Come posso personalizzare un metodo noto avvalendomi di un metodo “ad hoc”?
- Come posso sfruttare quanto detto per implementare un sistema di autorizzazione versatile e “non invasivo”?

Le risposte a queste domande verranno approfondite man mano.
Iniziamo con introdurre il concetto di “pacchetto di metodi”, più sinteticamente *Package*, che verrà ampiamente utilizzato.
Ogni archivio che deve far uso di *Stored Procedure* ovvero di *Trigger* avrà uno o più *Package* di metodi, divisi ragionevolmente per competenza, per ruolo, per mansione.
E' quindi intuitivo pensare che le *Stored Procedure* siano destinate ad essere collocate in *Package* distinti rispetto a quelli in cui si troveranno i *Trigger*.
Ancora: sia le prime che i secondi possono essere ipoteticamente divisi per unità informativa o più semplicemente per evento occorso sul quale scatenare il *Trigger*.
Se è vero quanto detto è altresì vero che *Stored Procedure* e *Trigger* di un archivio possono anche afferire tutti ad un solo *Package*.
Lo sviluppatore ha quindi il massimo grado di libertà nel definire il/i *Package* che preferisce.

Identificazione di un Package

Ciascun package è identificato per mezzo di una sequenza di etichette separate da punti ('.').
La composizione è la seguente:

ArchiveId.PackageId[.Sub-PackageId[.Sub-Sub-PackageId]].FunctionName

Vediamo ciascuna parte:

Parte	Interpretazione/Significato
ArchiveId	Consente l'identificazione dell'archivio presso il quale ve ricercato il *Package* da utilizzare/includere[11]. Ogni archivio ha una cartella nella quale i *Package locali* vengono collocati ma si possono utilizzare anche *Package* appartenenti ad altri archivi ovvero *Package di default*. Questi ultimi vengono collocati in un'apposita cartella dell'installazione del server *eXtraWay*. Qualora omesso[12] si ci si riferisce sempre all'archivio attuale. In seguito vedremo altre condizioni modificanti.
PackageId	Rappresenta l'identificazione del *Package* principale, che a sua volta può contenere dei *Sub-Package* a più livelli. Esso corrisponde anche al nome del file '**.lua**' che si contiene i metodi di questo *Package*
Sub-PackageId & FunctionName	Rappresentano l'ulteriore suddivisione, internamente al *Package* e quindi al file '**.lua**' in *Sub-Package* e *Funzioni*.

Con una sola sintassi identifichiamo: la cartella dove cercare i files `.lua` di nostro interesse, il nome del file da caricare e come interpretare il suo contenuto.

Ora alcuni esempi:

xdocwaydoc.decorator.tools.checkFolder
Questa sintassi ci indica di cercare nella cartella degli script dell'archivio noto col nome logico **xdocwaydoc**. Entro tale cartella cercheremo il file denominato **decorator.lua**. Quando avremo trovato tale file, al suo interno dovremo controllare la presenza di un metodo denominato **checkFolder** entro un *Sub-Pakcage* denominato **tools**
xdocwaydoc.trigger.doc.onLoad
xdocwaydoc.trigger.onLoad
Questw sintassi ci mostrano due esempi classici di dichiarazione di *Trigger*. Da essa sappiamo di dover cercare nella cartella degli script dell'archivio noto col nome logico **xdocwaydoc** e cercare al sui interno il file denominato **trigger.lua**. Quando avremo trovato tale file, avremo due possibilità: lo stesso medoto **onLoad**, tipico dei *Trigger* può essere ricercato e quindi invocato in due casi distinti, sia dichiarando entro il *Sub-Package* **doc**, sia entro il *Package* principale.

Un Package visto dall'interno

Come abbiamo detto, un *Package* altro non è che un file `.lua`.
Abbiamo visto come la denominazione completa del metodo che si intende utilizzare descriva come identificare tale file, ora abbiamo bisogno di vedere cosa ci dev'essere entro ciascuno di essi.
Ci sono due aspetti degni di attenzione: il caricamento di altri *Package*, alla stregua di librerie di funzioni, e l'esposizione dei *Sub-Packages* e dei metodi al chiamante, che si tratti di *eXtaWay Server* o di altro *Package* che sfrutta quello attuale come libreria.

Il caricamento di un altro *Package* come libreria di metodi si effettua evocando la funzione...
`xw.addPackage(“<Dichiarazione del package>”)`

...dove la Dichiarazione del package prende le mosse da quanto visto al paragrafo precedente, con alcune osservazioni/puntualizzazioni.
Possiamo infatti...

Indicare esplicitamente un *ArchiveId*	In questo modo indichiamo da quale archivio[13] desideriamo che il *Package* venga acquisito. **Nota: Se operando sull'archivio A carico script che sono propri dell'archivio B ed in essi le operazioni compiute non esplicitano a quale archivio esse si riferiscano, in quei casi l'archivio di riferimento è l'archivio A, quello dal quale ha preso origine il primo caricamento**.
Omettere l'*ArchiveId*	In questo modo indichiamo che il *Package* che si intende includere in quello attuale proviene dallo stesso archivio corrente
Indicare la costante '$'	In questo modo indichiamo esplicitamente che si desidera acquisire il *Package* non da quest'archivio ne da qualsiasi altro bensì dalla cartella di sistema ove vengono distribuiti script lua di utilità generale. Tale cartella è la cartella `script` parallela alla cartella dei binari del server *eXtraWay*.
Indicare la costante '*'	In questo modo indichiamo esplicitamente che si desidera acquisire il *Package* dall'archivio corrente ma, qualora non risulti presente alcun *Package* identificabile col nome dato, esso viene ricercato ed acquisito dalla cartella di sistema, come indicato al punto precedente.

Veniamo ora alla parte più importante e significativa: la dichiarazione, all'interno del *Package* dei *Sub-Packages* e dei metodi esposti.
Questo si compie per mezzo della dichiarazione di una tabella di valori secondo la sintassi Lua.
Torniamo a prendere i 3 esempi di cui sopra, quelli che si riferiscono a queste dichiarazioni:

```
xdocwaydoc.decorator.tools.checkFolder
xdocwaydoc.trigger.doc.onLoad
xdocwaydoc.trigger.onLoad
```

Quello che abbiamo già visto è che troveremo i file `decorator.lua` e `trigger.lua` nella cartella degli script dell'archivio `xdocwaydoc`.
Arricchiamo il nostro esempio con ulteriori dettagli.
Come vedremo in seguito, i *Trigger* hanno parametri predefiniti, ma ciò nonostante essi vanno comunque dichiarati.
Per le *Stored Procedure* è invece ovvio che essi vanno sempre espressi puntualmente.

Andiamo per ordine.
La nostra *Stored Procedure* denominata `checkFolder` appartenente al *Sub-Package* denominato `tools` e prevede alcuni parametri in ingresso ed uscita.
In ingresso avremo il codice del fascicolo, detto `codFolder`, ed un `flag` che ci indica in che modo dobbiamo fare il test. Il `flag` può valere “weak” o “strong” ad indicare la modalità. Se non indicato il suo valore di default sarà “stong”.
La funzione torna due parametri: il primo dice se l'operazione è andata a buon fine con un valore numerico e si chiama `result`, il secondo è una descrizione verbosa delle operazioni svolte e si chiama `details`.
Nel nostro file `decorator.lua` avremo la seguente dichiarazione

decorator.lua

```
xw.addPackage("$.xwTools") ;

local myFunctionToCheckFolders = function(codFolder, flag) -- Input Parms, no way to define
defaul values
   local result = 1 ;
   local detail = "" ;
   -- Omitting real code...
   return result, detail ; -- Output Parms
end

decorator = { -- Package
   tools = { -- Sub Package
      checkFolder = myFunctionToCheckFolders, -- Exposing the internal (local) function with
a name
      checkFolder_signature = { -- Telling how to call and what to expect
         inParms = {"codFolder", "flag"}, -- Input parms declaration
         default = {"", "strong"}, -- Default Values declaration
         outParms ={"result", "detail"} -- Output Parms declaration
      }
   }
} ;
```

Nel precedente esempio vediamo qualcosa di ben preciso.
Lo script inizia includendo un *Package* denominato `xwTools` ed appartenente alla directory di sistema, sono funzionalità già predisposte da 3D e distribuite con il resto dell'estensione Lua.
In seguito si definisce una funziona locale, quindi non accessibile dall'esterno se non espressamente esposta, che si chiama `myFunctionToCheckFolders`. La funzione prevede due parametri ma dalla dichiarazione della funzione non si hanno ulteriori dettagli. La funzione, a sua volta, torna due valori.
In coda allo script si dichiarano un sistema di `tables` così costituito:

- La prima table in assoluto dev'essere l'ID del package;
- Entro tale table possono essere impostati direttamente dei metodi o degli ulteriori *Sub Packages* per un numero indefinito di livelli;
- Quando si giunge alla dichiarazione del metodo esposto si accoppia un nome “esterno” al nome “locale”. Questo consente ad esempio di avere un'unica funzione che assolve a più compiti o comunque tutelarsi, qualora non si usino appropriatamente i modificatori `local`, che dall'esterno non verrà mai evocato un metodo al posto di un altro;
- Parallelamente a ciascun metodo dichiarato si costituisce un'ulteriore sotto-tabella finalizzata a dichiarare i parametri di input (col loro default) e di output.
  In particolare è bene sapere che:
  - Tutti i parametri (inParms, default, outParms) possono essere omessi: una funzione può non prevedere un input[14] o non fornire un output[15].
  - Solo le funzioni che hanno dei parametri di input possono avere un default. Esso può quindi essere omesso in molteplici casi. Se però va indicato, essendo posizionale, si devono indicare tutti i valori (anche se nulli ovvero corrispondenti a stringa vuota). Nel nostro esempio il secondo parametro ha un valore di default, ma la dichiarazione di tali valori mi impone di indicare qualcosa (vuoto) anche per il primo di essi.

Procediamo con un altro esempio: i *Trigger*

triggerSample.lua

```
xw.addPackage("$.xwTools") ;

local docOnLoad = function(doc, num)
   local newDoc = "" ;
   -- Omitting real code...
   return newDoc ;
end

local anyOnLoad = function(doc, num)
   local newRecord = "" ;
   -- Omitting real code...
   return newRecord ;
end

trigger = {
   doc = {
      onLoad = docOnLoad, -- This is for ''doc'' Information Units only
      onLoad_signature = {
         inParms = {"doc","num"},
         outParms = {"doc"}
      }
   },
   onLoad = anyOnLoad, -- For Any other Information Unit
   onLoad_signature = {
      inParms = {"doc","num"},
      outParms = {"doc"}
   }
} ;
```

Anche in questo caso abbiamo incluso il *Package* di sistema noto come `xwTool`, poi abbiamo implementato due metodi da evocare in corrispondenza dell'evento *onLoad* avendo cura di distinguerli. Uno di essi viene esplicitamente evocato quando l'evento si verifica su un record di tipo `doc` mentre l'altro verrà richiamato lo stesso evento che dovesse occorrere su qualsiasi altro tipo di Unità Informativa.
Il meccanismo della *signature* è analogo ma ci sono importanti particolari che vedremo in seguito quando si scenderà nel dettaglio del trattamento dei *Trigger*.

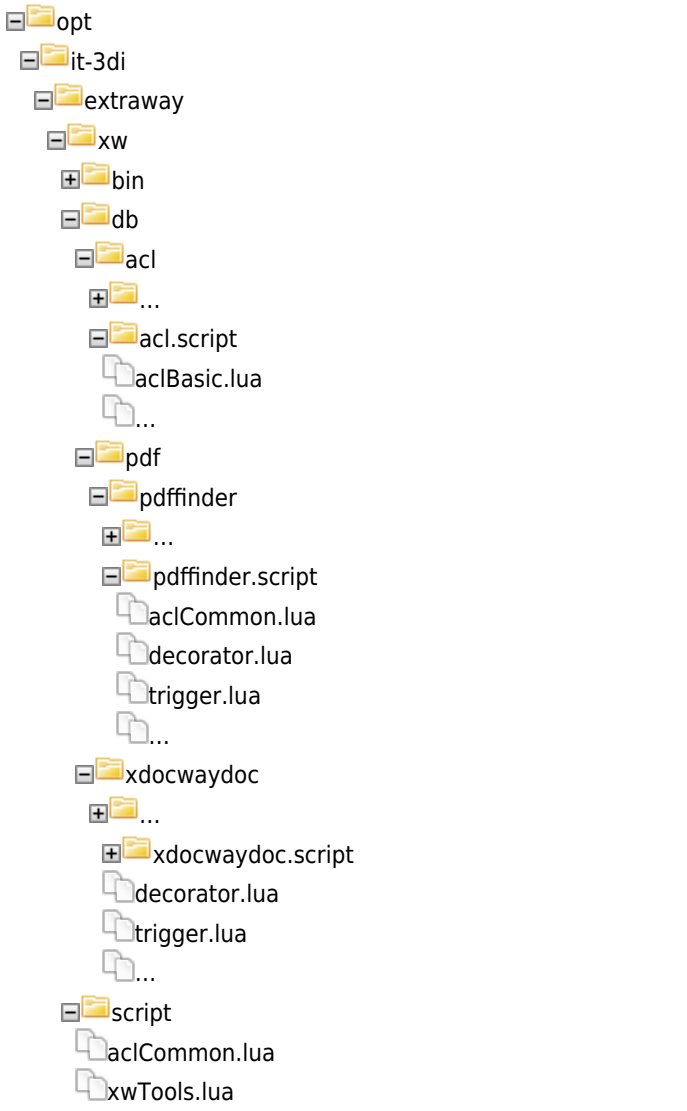
Un Package visto dal disco

Gil esempi fatti sino ad ora ci dicono come identifichiamo i file contenenti gli script lua e come ne trattiamo il contenuto.
Ora vediamo un po' meglio come distribuiamo su *file system* i nostri files.

Continuiamo ad immaginare di avere un archivio `xdocwaydoc` il quale ha propri *Package* che sfruttano quelli di sistema.
Immaginiamo inoltre di avere altri archivi: un archivio `pdffinder` ed un archivio `acl` utilizzato da entrambe.
Partiamo dal presupposto che il server sia installato in `/opt/it-3di/extraway/xw/bin`.
Per identificarli tutti usiamo il file `xw.ini` nell'apposita sezione...

```
[Archivi]
pdffinder=/opt/it-3di/extraway/xw/db/pdf/pdffinder/pdffinder
```

La sezione dichiara solo un archivio (in quanto in collocazione non standard) mentre gli altri sono nella collocazione di default per il proprio nome.
Ecco la dislocazione dei files



Ciò che notiamo immediatamente è che entrambe gli archivi hanno il proprio `decorator.lua` ed il proprio `trigger.lua` ed essi sono comunque distinti in quanto l'*ArchiveId* sarà sufficiente ad identificare correttamente quale *Package* stiamo per usare.
Notiamo anche che lo stesso script `aclCommon.lua` è presente tanto nell'archivio `pdffinder` che nel pacchetto delle librerie di sistema.
Perché fare una cosa simile?
Una possibile spiegazione aiuta a comprendere i meccanismi di inclusione degli script.
Supponiamo che in entrambe gli archivi `pdffinder` e `xdocwaydoc` i *Trigger* debbano fare uso di script di autenticazione.
È buona prassi usare interfacce e metodi omogenei quindi nell'archivio `pdffinder` si è provveduto a creare un *Package* con la stessa denominazione di quello di sistema.
Se entrambe i file `trigger.lua` dei due archivi includono...

```
xw.addPackage("*.aclCommon") ;
```

... il trigger di `xdocwaydoc` caricherà quello della directory di sistema mentre quello di `pdffinder` caricherà il proprio in quanto omonimo ma presente nel percorso di ricerca degli script in via privilegiata.

Un Package nella configurazione d'archivio

Qualsiasi archivi può prevedere l'uso di *Trigger* e *Stored Procedure*.
I due strumenti sono del tutto distinti ma accomunano molti aspetti inerenti la configurazione.
Come abbiamo visto prima, che si tratti degli uni o degli altri, ciascun *Package* deve citare per mezzo di una tabella costituita come mostrato tutti i metodi che espone e, per ciascuno di essi, tutti i parametri in ingresso ed in uscita nonché gli eventuali valori di default.
Risulta intuitivo che le *Stored Procedure* non richiedano particolare configurazione in quanto esse vengono chiamate esplicitamente sulla base del loro nome, completo dell'identificazione del *Package*, e dei parametri dichiarati.
Evocando con un comando XML una *Stored Procedure*...

cmd.xml

```
<?xml version="1.0"?>
<cmd c="0" stored="xdocwaydoc.decorator.checkFolder">
<myParam>This is a description</myParam>
</cmd>
```

Se prendiamo questo esempio e lo confrontiamo con quanto visto nei precedenti paragrafi notiamo che in questo caso il parametro `codFolder` previsto e richiesto, è assente. Non avendo un proprio default, tale parametro viene passato in forma di “stringa vuota”. Discorso diverso per il parametro `flag` che è sì assente ma per esso verrà usato il default descritto nello script lua.
Il parametro `myParam` non è citato tra quelli previsti e non causa alcun problema alla chiamata del metodo `xdocwaydoc.decorator.checkFolder` ma viene semplicemente ignorato.

Discorso differente va fatto per i *Trigger*.
Lo scopo di un *Trigger* è ovviamente di intervenire automaticamente in determinate situazioni e questo si presta ad un'importante applicazione oltre agli scopi applicativi: la sicurezza e l'autorizzazione.

Abbiamo detto che dal *Trigger* ci si aspetta un intervento automatico. Diversamente dalle *Stored Procedure* quindi non avremo un'invocazione consapevole bensì la necessità di sapere a priori quale sia il *Package* da interrogare.
Perché questo avvenga si interviene nel file di configurazione dell'archivio introducendo una configurazione puntuale.

```
   <profile type="trigger.user" value="trigger"/>
```

L'esempio mostra che il *Package* che corrisponde ai *Trigger* d'utente si chiama, appunto, `trigger`.
Il trigger che si va a caricare quindi sarà specifico dell'applicazione che procederà con la ricerca del file `trigger.lua` nella cartella degli script dell'archivio.
In seguito vedremo, per ciascun evento possibile, come si devono chiamare i metodi corrispondenti ai diversi eventi.
Una volta identificato il *Package* nel quale cercare i metodi da evocare il server seguirà questo criterio per la ricerca suddetta:

- Ricerca di `.<Trigger Package>.<Unità Informativa>.<evento>`.
  Nel *Package* indicato si cerca la presenza di un *SUb-Package* il cui nome deve corrispondere a quello del *Primary Node Element* sul quale si sta compiendo l'operazione.
  In tale tabella, poi, si accede al metodo che porta il nome dell'evento;
- Ricerca di `.<Trigger Package>.<evento>`.
  Qualora il test di cui al punto precedente non rilevasse una funzione valida, si procede alla ricerca di una funzione dichiarata per l'evento ma non riferita ad un *Primary Node Element* specifico.
  In presenza di una simile funzione essa si applica quindi a tutti i casi in cui non è stata esplicitata una funzione “ad hoc”.
  Questo approccio si applica ovviamente anche agli eventi che non prevedono l'indicazione specifica di una *pne*, come la ricerca.

In questo modo, quindi, il server sa identificare perfettamente, per ciascuna *Primary Node* e per ciascun evento, quale sia il metodo da evocare.
Quando per un determinato evento non esiste alcun *Trigger* identificabile con o senza *Primary Node Element* il server non esegue alcuna operazione. Come abbiamo già detto il server mette a disposizione una serie di parametri (indicati in seguito) ma i metodi del *Package* dei *Trigger* possono prevederne solo un sottoinsieme.

Quanto detto per i *Trigger* d'utente si può applicare anche ad altri *Trigger* che però assumono il ruolo di verifica di autorizzazione.
In presenza di un sistema di diritti, gestito per mezzo di un qualsivoglia *provider*[16], si possono prevedere dei *Trigger* finalizzati, ad esempio:

- Impedire inserimenti e modifiche di record per i quali non si gode dei necessari diritti;
- Limitare la visibilità dei record filtrando opportunamente tutte le selezioni compiute dall'utente;
- Filtrare il contenuto di un record in sede di visualizzazione per consentire che l'utente prenda visione solo delle parti di sua competenza;
- Altro...

Il meccanismo è del tutto identico al precedente e prevede ovviamente la sua configurazione.

```
<profile type="trigger.auth" value="trigger"/>
<profile type="trigger.auth.mode" value="weak"/>
```

Come vediamo la configurazione necessaria è del tutto simile alla precedente e differisce solo per due dettagli:

1. La voce di configurazione è di tipo `trigger.auth` anziché `trigger.user`. I due *Packages* convivono senza infastidirsi recirocamente e questa modalità consente, in futuro, di introdurre altri filoni di *Trigger*;
2. Il *Trigger* d'autorizzazione prevede una modalità che può essere `weak` o `strong`.
   Come abbiamo visto in precedenza, l'omissione di un *Trigger* che risponda ad una particolare condizione (un evento su un *pne*) porta il server ad ignorare l'operazione.
   Nell'ambito della sicurezza questo comportamento potrebbe essere pericoloso e quindi si impone la dichiarazione della modalità così che nulla sfugga al controllo incrociato del *Server eXtraWay* e dello Sviluppatore.
   Vediamo come:
   1. In caso di modalità `weak` l'assenza di un metodo per un dato evento viene tollerata e non conduce ad alcun errore. Si ha, in sostanza, il comportamento di *Trigger* d'utente;
   2. In caso di modalità `strong` l'assenza di un metodo per un dato evento (che sia esplicitamente legato alla *pne* o che sia generico) comporta una condizione d'errore. Il server impedisce l'operazione e produce un codice d'errore specifico;
   3. In caso di mancata dichiarazione nel file di configurazione della modalità suddetta si assume che essa sia sempre `strong`.

Nota:Se si evoca una *Stored Procedure* con più parametri di quelli citati dalla sua *signature* i parametri inutilizzati vengono semplicemente ignorati.
Un discorso analogo va fatto anche per i *Trigger*. In seguito vedremo gli elenchi con i parametri che il server predispone prima di compiere una chiamata ma lo sviluppatore che realizza un *Trigger* ha facoltà di utilizzare realmente un minor numero di essi. Rimane il vincolo, in ambo ci casi, che i parametri realmente richiesti dal metodo Lua siano effettivamente inviati. Per analogia, il chiamante ha facoltà di ignorare uno o più degli eventuali parametri di ritorno.

Eventi Triggers

Al momento in cui questa documentazione viene redatta esistono i seguenti eventi (con relativi parametri di input ed eventuale output):

Evento		**Descrizione**
beforeSave (Autoritativi: preBeforeSave e postBeforeSave)		Intervento sul record che si sta provvedendo ad inserire/modificare prima che il salvataggio abbia luogo e prima che l'archivio venga bloccato
	pne	Il *Primary Node Element* del record da salvare
	doc	Il record che si sta per salvare
	num	Il numero fisico del documento che si sta per salvare, in caso di inserimento il valore è '0'
		*Torna un '**doc**' col quale modificare l'originale. Se non viene tornato un valore valido (o stringa vuota) il documento originario rimane invariato.*
onExtend		Intervento sul record che si sta provvedendo ad inserire/modificare prima di completare il salvataggio per completare il record con una parte calcolata *ad-hoc* equivalente a quella realizzata con estensione `Xslt`
	pne	Il *Primary Node Element* del record da salvare
	doc	Il record che si sta per salvare
		*Torna un '**doc**' da aggiungere all'originale. Se non viene tornato un valore valido (o stringa vuota) non viene applicata alcuna estensione.*
afterSave afterIndex		Intervento sul record che si appena provveduto a salvare/rimuovere, prima o dopo che gli indici siano creati/aggiornati.
	pne	Il *Primary Node Element* del record appena salvato
	oldDoc	Il record com'era prima di essere salvato. Se il valore è una stringa nulla, allora si tratta di inserimento
	newDoc	Il record così come lo si è appena salvato. Se il valore è una stringa nulla, allora si tratta di cancellazione
	num	Il numero fisico del documento che è appena stato salvato, in caso di inserimento il valore è quello attribuito in fase di salvataggio
		Non ha output
beforeTitle[17)]		Intervento sulla regola di composizione del titolo per personalizzarla a piacimento.
	rule	La regola di composizione del titolo che ci si appresta ad utilizzare

Evento		Descrizione
		Produce in output una nuova '***rule***' *ovvero stringa vuota ad indicare che la regola rimane immutata.*
afterTitle[18]		Intervento sulla stringa del titolo appena composta.
	num	Il numero fisico del documento cui si riferisce il titolo
	title	Il titolo calcolato per tale documento
	rule	Regola di composizione del titolo utilizzata
		Produce in output un nuovo '***title***' *ovvero stringa vuota ad indicare che il titolo rimane immutato.*
onLoad **onLoadLocked (Autoritativi: preOnLoadLocked, preOnLoad, postOnLoadLocked, postOnLoad)**		Intervento sul record che si sta provvedendo a caricare. Si noti che quest'attività viene compiuta sempre, in quanto *Trigger*. Questo può avere impatti considerevoli dal punto di vista prestazionale e può comportare il rischio di *infinite loop*. Per questa ragione si raccomanda di far uso di questo *Trigger* con estrema cautela e prediligere l'uso di una *Stored Procedure* qualora si intenda dare a questa funzionalità una finalità squisitamente decorativa.
	pne	Il *Primary Node Element* del record che si procede a caricare
	doc	Il record che è stato appena caricato
	num	Il numero fisico del documento che si carica
		Torna un '***doc***' *col quale modificare l'originale. Se non viene tornato un valore valido (o stringa vuota) il documento originario rimane invariato.*
onTitle **onSort**		Intervento sul record che si sta provvedendo a caricare per *scopi interni*. Quest'attività, pur trattandosi di un trigger, viene di fatto eseguita solo quanto necessaria, vale a dire per produrre la *cache titolo* per i titoli stessi ovvero per l'ordinamento. I due triggers vengono quindi evocati solo se la *cache del titolo* manca o se cerco di fare un titolo ovvero ordinare per qualcosa che non è previsto nella cache.
	pne	Il *Primary Node Element* del record che si procede a caricare
	doc	Il record che è stato appena caricato
	num	Il numero fisico del documento che si carica
		Torna un '***doc***' *col quale sostituire l'originale per comporre un titolo o compiere un ordinamento. Se non viene tornato un valore valido (o stringa vuota) si fa uso del documento.*
onWatchDoc		Operazione conclusiva sull'acquisizione record via *WatchDoc*.
	file	Il nome del file elaborato da *WatchDoc* (comprensivo dell'estensione di lavoro `.wkr`).
		Attualmente non prevede un valore di ritorno.
onSearch (Autoritativi: postOnSearch)		Intervento sulla frase di ricerca. Si presta particolarmente per imporre filtri di selezione legati ai diritti dell'utente.
	query	L'espressione di ricerca da risolvere
		Torna un '***newQuery***' *col quale modificare la ricerca. Se non viene tornato un valore valido (o stringa vuota) la ricerca originaria rimane invariata.*

Si ricorda che una *Stored Procedure* così come un *Trigger* può evocare altri *Trigger* con semplici operazioni di ricerca, caricamento, inserimento e modifica record.
Chi realizzerà i *Trigger* deve tenere conto del rischio di causare una eventuale condizione di *loop* o un effetto a catena.

"Reference" dei metodi e dei tipi disponibili

La stesura di una *Stored Procedure* ovvero di un *Trigger* in Lua può essere effettuata avvalendosi dei seguenti metodi e tipi complessi.

Iniziamo dalla descrizione dei tipi complessi che si sommano ai tipi standard di Lua.

Tipo	Finalità
xwLuaRecord_t	Rappresenta un singolo record di un archivio eXtraWay. Esso può essere sia un record esistente al quale si accede sia un record del tutto nuovo che potrà essere salvato nell'archivio.
xwLuaRecordSet_t	Rappresenta un insieme di `xwLuaRecord_t`, tipicamente esito di un'operazione di selezione sull'archivio eXtraWay.
xwLuaNode_t	Dal momento che i `xwLuaRecord_t` sono degli XML essi sono composti da *nodi* di un *DOM*. Il `xwLuaNode_t` rappresenta proprio uno di tali nodi, nella più generica delle accezioni possibili.
xwLuaNodeSet_t	Analogamente ai casi precedenti, un `xwLuaNodeSet_t` rappresenta un insieme di `xwLuaNode_t`, normalmente frutto dell'applicazione di un *XQuery* sul record.
xwLuaStringBuffer_t	Rappresenta un unità di storage che può essere in ram o su file.
xwLuaMap_t	Mappa associativa utilizzabile per chiamate complesse a funzioni del server.

Ciascuno di tali tipi evoluti ha metodi propri, così come esistono metodi generali, riferiti tipicamente a chiamate dirette al server eXtraWay.

Evocare un metodo di un tipo evoluto prevede una sintassi lievemente differente da quella da adottare per evocare un metodo generico.
Mentre il metodo generico prevede la sintassi...

```
<namespace>.<metodo>(…)
```

...il metodo evoluto richiede la forma...

```
<tipo evoluto>:<metodo>(…)
```

...per compiere una chiamata analoga.

Si suggerisce quindi la massima attenzione nell'uso di `.` e `:`.

Di seguito il *reference* di dettaglio dei metodi con una notazione Lua.

La Reference

Per avere una **Reference** sempre puntualmente aggiornata essa viene generata dinamicamente dal codice LUA implementato per *eXtraWay*.
Per tale ragione tale documentazione non è direttamente presente in queste pagine **ed è invece disponibile in una collocazione separata.**

1) Tipicamente in sede di salvataggio dei record, di caricamento, di indicizzazione e di elaborazione dei titoli
2) in precedenza detta liblua
3) in precedenza detta libluafunc
4) \r\n
5) Portata anche nella precedente versione
6) Sino ad ora, per mantenere un comportamento coerente con altri `plug-in`, il server teneva traccia dei record bloccati dall'utente durante l'esecuzione di uno script LUA e provvede a sbloccarli tutti una volta che lo script termina
7) che si chiama anch'essa `liblua.so` ma va collocata in `libs`
8) xwLuaStringBuffer_t
9) Richiede Server eXtraWay 24.12.0 o superiore
10) In assenza di un metodo specifico per l'unità informativa dichiarata si “ricade” sul metodo generico
11) Si veda, nella *Reference*, il comando `xw.addPackage()`
12) Ma non si può e non si deve omettere il punto che lo speara dal *Package*
13) sarà il server con la propria configurazione a risolvere l'ID in un percorso valido
14) ma in quel caso avrà almeno un output
15) ed in questo caso avrà certo un input
16) LDAP, lo stesso ACL di 3D Informatica ovvero altri strumenti
17) , 18) Dalla versione 25.5.0 del Server eXtraWay