Manual del usuario de OIS

Versión 02.03

Object's Indexed Serialiation

Un sistema jerárquico de almacenamiento de objetos TOL de estructura arbitraria basado en ficheros binarios

Bayes Decision S.L.

Gran Vía, 39, 5ª planta
28013 Madrid (España)
Tel. (34) 915327440
Fax. (34) 915322636
www.bayesforecast.com
www.tol-project.org

Contenido

Resumen

La serialización indexada de objetos TOL pretende dotar al lenguaje de programación TOL[1] de un mecanismo de almacenamiento externo permanente de funciones y objetos TOL previamente evaluados, que posteriormente pueden ser recargados de forma independiente en ulteriores sesiones de TOL.

En este documento se da un diseño preliminar de un sistema completo y un presupuesto de desarrollo de un prototipo con las funciones básicas mínimas.

¡Atención!

Documento en construcción

Uso obligatorio de casco y botas

[1]TOL: Time Oriented Language, un lenguage interpretado de programación open-source cuya página web es [http://www.tol-project.org||TOL-project] y que ha desarrollado por el grupo [http://www.bayesforecast.com||Bayes]

Copyright	2006, ***Bayes Decision S.L.***
Título	Manual del usuario de OIS
Asunto	Object's Indexed Serialiation
Categoría	Documento técnico
Archivo	C:\projects-c++\bayes\tol-project\tol\doc\OIS\OIS.ManualDeUsuario.odt
Edición	13/05/06 17:20:25
Claves	almacenamiento, indexación, objetos
Distribución	Pública

1. Descripción y requerimientos

Un OIS es un contenedor en un soporte permanente de un conjunto TOL y todos sus objetos descendientes de forma jerárquica, que se puede cargar total o parcialmente en ulteriores ejecuciones de TOL, cuando se desee, en cualquier ámbito de ejecución y de forma inmediata y transparente.

Aunque en prinipio el concepto de OIS es independiente del tipo de soporte hardware/software utilizado, si no se especifica lo contrario nos referiremos a OIS en un sistema de ficheros binarios agrupados en un directorio dado.

Además de las características típicas de un objeto TOL, como el tipo de datos, el nombre, la descripción o la expresión TOL correspondiente, se almacenarán también, cuando sea posible y de forma opcional, sus **representaciones serializadas**, es decir, precalculadas, para mayor eficiencia en las recargas. Una vez cargados en una sesión posterior de TOL, estos objetos además de visualizables en el interfaz, serán referenciables en cualquier expresión como cualquier otro, y de forma totalmente transparente para el usuario.

2.Posibles utilidades y modos de empleo

2.1 Librerías TOL de módulos precompilados

Una de las utilidades más importantes de OIS es la de empaquetar grandes librerías de funciones y objetos cuyo código fuente no se actualize con excesiva frecuencia, como por ejemplo la propia initlibrary de TOL o sistemas como SADD en lo que llamaremos **librerías precompiladas**. Con esto se conseguirá un mejor manejo de las mismas, aprovechando las características de organización, compactación y carga bajo demanda de objetos que porporciona OIS.

Se mantiene la **integridad referencial a nivel de fichero** detectando cuando el OIS depende de un fichero TOL que ha sido modificado en tamaño, o contenido desde el momento de su construcción procediendo a su reconstrucción inmediata si ha habido cambios. Para ello los módulos OIS almacenan todos los ficheros de código fuente en un subdirectorio `._tol_source_` en el que cada fuente se guarda con su path absoluto transformado para que sea un camino válido interno.

Si se producen errores durante la compilación del módulo no se creará la imagen OIS. También es necesario que un OIS detecte dependencias de otros módulos OIS recursivamente[2]

[2]Aún sin implementar

Es recomendable incluir básicamente estructuras, funciones y sólo los objetos globales de configuración indispensables, y dejar el resto de objetos para los siguientes apartados.

2.2 Almacén estático de objetos TOL

La principal utilidad de TOL es la de permitir almacenar cualesquiera objetos de una forma eficaz tanto en tiempo de almacenamiento y recuperación como de espacio requerido. Este tipo de imágen OIS no precisa de ningún tipo de sistema de integridad referencial pues se trata de una foto fija.

Por ejemplo, para guardar modelos organizados por sesión de estimación y output se creará un directorio para cada sesión, a la que llamaremos versión por homogeneidad de nomenclatura, y dentro de cada uno habrá un directorio OIS para cada output. También se puede complicar está estructura haciendo que haya nodos jerárquicamente organizados unos dentro de otros.

Para que un sistema así sea eficaz es necesario que permita la carga parcial de subconjuntos de datos especificados por el usuario[3], así como algún método de navegación interna[4] para la exploración del contenido de un OIS.

2.3 Caché de objetos TOL

También se puede utilizar un OIS como simple **método de caché**, para ahorrar memoria y CPU en procesos masivos, almacenando resultados parciales para la ejecución de diferentes fases de procesos que no se puedan ejecutar de un sólo trazo.

Además del sistema de integridad referencial usado en las librerías precompiladas, es necesario que ampliar el control de actualizaciones para asegurar que lo que se carga del OIS es lo que se esperaba. Para ello el usuario deberá especificar dichas condiciones adicionales.[5]

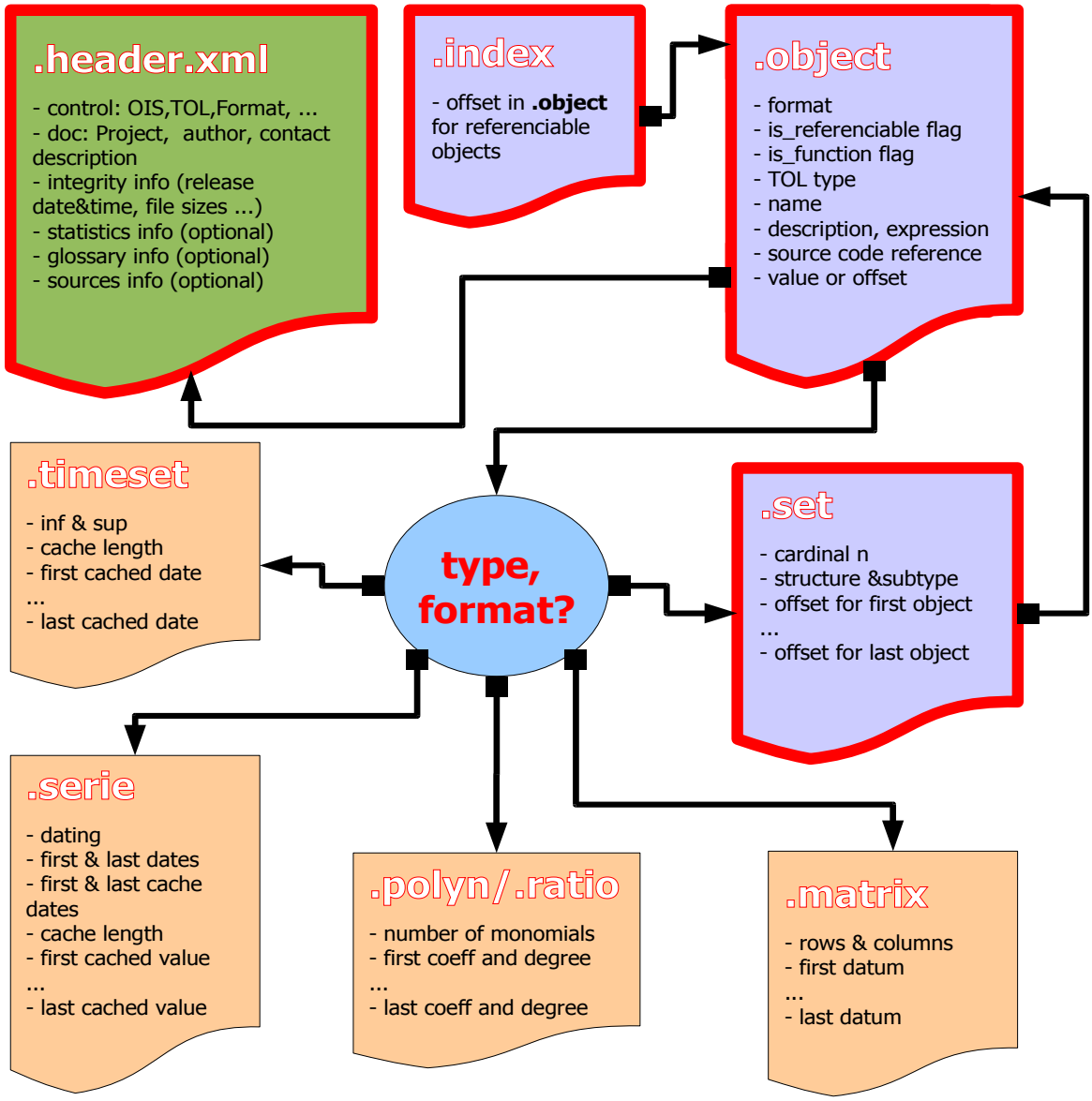
Un caso típico es aquel en el que intervienen series infinitas que se almacenan entre las fechas por defecto. Si estas cambian, lógicamente se ha de rehacer la imagen OIS. Otro caso habitual es aquél en el que los datos almacenados se construyen a partir de objetos cargados de la base de datos, lo cual hace imposible detectar si ha ocurrido algún cambio. En estos casos el criterio de reconstrucción puede ser externo como un apéndice del proceso de actualización de datos en el servidor o bien se puede hacer de forma periódica, si se conoce una periodicidad mínima entre actualizaciones.

[3]Implementado un prototipo en pruebas
[4]aún está en desarrollo
[5]De momento se ha de hacer manualmente con código TOL escrito ad-hoc en cada caso para tal fin

3. Estructura de un OIS

Un OIS en soporte de ficheros binarios se compone de un conjunto de archivos que contienen las expresiones, datos y relaciones de los objetos TOL almacenados.

Ilustración 1: Esquema del diseño de un OIS binario



Todos los archivos se ubican en un mismo directorio y tienen un nombre que empieza por punto, al que llamaremos extensión del OIS.

Los archivos con borde rojo resaltado en el esquema de la Ilustración 1 son obligatorios y el resto opcionales. Las flechas dirigidas indican que la entidad de origen contiene referencias a la de destino.

Todos los ficheros excepto los del área auxiliar, tendrán un formato binario inmodificable e independiente de la máquina y el sistema operativo, para poder ser compartidos desde cualquier plataforma sin necesidad de realizar ningún proceso de importación ni exportación.

3.1 Área auxiliar

Incluye toda la información auxiliar necesaria para la interpretación y el manejo interno y documental. Son los archivos en verde del esquema de la Ilustración 1:

1. `.header.xml` : Ha de existir obligatoriamente este archivo de cabecera en el que se almacenarán en formato XML las siguientes secciones
    1. Una **sección de control** con la información necesaria para descodificar el resto de ficheros, mediante el número de versión de TOL y OIS con el que ha sido creado.
    2. Una **sección documental** con información sobre el autor y el proyecto en el que se enmarca y una descripción del contenido que permita hacer búsquedas.
    3. Una **sección de integridad referencial** para detectar problemas debidos a la manipulación indebida de los ficheros o a cualquier tipo de error. Como mínimo se debe hacer constar la fecha y hora de creación del OIS y el tamaño de cada uno de los ficheros que lo integran.
    4. Una **sección opcional de información estadística** y de control sobre el número y tamaño de los de objetos de cada tipo de datos
    5. Una **sección opcional de glosario de términos** que contenga el número de apariciones de todos los nombres de variables y palabras utilizadas en las descripciones y expresiones. También se pueden incluir grupos de palabras consecutivas de hasta un cierto orden para búsquedas más complejas.
    6. Una **sección opcional de fuentes** conteniendo una lista de referencias a los archivos TOL de código fuente (.tol, .bst, .bdt, ...) y ubicaciones históricas, por ejemplo algo estilo CVS. Esto puede ser muy útil si se requiere indagar en el proceso de desarrollo de un proyecto. Incluso puede ser interesante incluir el contenido mismo de dichos ficheros fuente[6].

3.2 Área de declaración

Incluye los ficheros de declaración de objetos y referencias de TOL. Son los ficheros en añil del esquema de la Ilustración 1:

[6]Lo normal es que el tamaño del código sea despreciable en comparación con el de los datos así que normalmente no supone un grave problema.

2. `.object` : Ha de existir obligatoriamente un fichero de declaración de los objetos contenidos, siendo el primero de ellos el conjunto raíz del OIS, e incluyendo para cada objeto lo siguiente:
   1. La información de indexación, que determine si es global o local, función u objeto, y de qué tipo, así como su nombre.
   2. La información adicional como su expresión y descripción, si es que tiene.
   3. Información para la construcción de los datos asociados, bien directamente en los casos de los tipos escalares (Real, Complex, Text, Date, Code), bien su serialización en los ficheros de caché correspondientes (.Set, .Serie, .TimeSet, .Matrix, .Polyn, .Ratio)
3. `.oisref` : Ha de existir obligatoriamente un fichero, posiblemente vacío, con las direcciones de los objetos accesibles por el lenguaje TOL, es decir, los objetos con nombre declarados a nivel global, o mejor dicho al mismo nivel o superior a aquél en el que se construyó el OIS. Las direcciones vendrán dadas por su posición (offset) en el fichero .object
4. `.set` : Ha de existir obligatoriamente un fichero de definición de la jerarquía de pertenencia de los objetos en los conjuntos anidados, mediante una lista de direcciones de sus elementos dadas por su posición (offset) en el fichero .object
5. `.hrchyDetail, .hrchyOffset, .hrchyOrder` : Puede existir de forma opcional este conjunto de ficheros que almacenan una representación de toda la jerarquía de conjuntos de .set de forma accesible permitiendo posteriormente la carga parcial de una rama del OIS.

3.3 Área de definición

Incluye la definición de los datos propiamente dichos. Cada uno de los registros de estos archivos permite la especialización del formato de almacenamiento de forma particular para cada objeto. Son los archivos en naranja del esquema de la Ilustración 1:

6. `.serie` : para contener sus definiciones, incluyendo el fechado, las fechas inicial y final, y, si la serie necesita caché por razones de eficiencia o porque es una serie primaria sin método de cálculo, también las fechas de inicio y fin de la caché, el número de datos y una lista ordenada de ellos.
7. `.timeset` : para contener sus cachés por razones de eficiencia, incluyendo las fechas de inicio y fin de la caché, el número de fechas y una lista ordenada de ellas.
8. `.matrix` : para contener las matrices, incluyendo el número de filas y columnas y la lista de datos. Para matrices especiales es muy conveniente contar con métodos particulares de representación.
9. `.polyn` : de los poinomios se guarda el número de monomios y una lista ellos definidos como pares (grado, coeficiente).

10. `.ratio` : con el mismo formato que el .polyn pero con un par de polinomios consecutivos, correspondientes al numerador y denominador.
11. `.code` : De las funciones se almacena el árbol resultado del parser por lo que se puede hablar de código precompilado.
12. `.str.<struct_name>`[7] : Aunque en el esquema anterior no se representan por simplicidad en la exposición, es conveniente contar con un tipo de almacenamiento especialmente diseñado para conjuntos con estructura de tabla, en el que cada elemento es un conjunto con la misma estructura struct_name. Para los tipos de datos escalares se incluirá directamente el valor del objeto, por lo que estos pierden los atributos TOL como el nombre y la descripción, mientras que para el resto de tipos se incluirá el offset al archivo .object, igual que en el archivo .set. Existirá por tanto un archivo distinto para cada Struct TOL para el que existan conjuntos definidos con dicha estructura. Estos archivos no son imprescindibles pero pueden ser muy útiles para comprimir datos masivos y aligerar el tratamiento en TOL de grandes cantidades de datos sin las complicaciones de manejo añadidas por las bases de datos.

4. Crecimiento de OIS

OIS debería tener una serie de dispositivos que permitieran un uso eficaz y sencillo que permita aprovechar al máximo toda la potencia de OIS

4.1 Sistema gráfico de previsualización en TOLBase

Pendiente de diseño e implementación

4.2 Sistema de búsqueda y consulta externa

Pendiente de diseño e implementación

4.3 Sistema de generación automática de documentación

Pendiente de diseño e implementación

4.4 Sistema de ubicación de repositiorios vía web

Pendiente de diseño e implementación

4.5 Ois.Remote

Permitiría el intercambio de imágenes OIS entre procesos TOL en diferentes máquinas. Pendiente de diseño e implementación

[7]No está claro que este apartado se deba desarrollar pues puede suponer demasiada complicación.

5. Organización de sistemas OIS masivos

Un OIS ha de ser comprimible y empaquetable de una forma natural pues puede alcanzar tamaños más que respetables, pero ha de ser consultable en su estado original de forma transparente para el usuario.

Para mayor comodidad de manejo, un conjunto de OIS's se puede empaquetar en un sólo archivo con extensión compuesta `.oza` (**Ois Zip Archive**)[8] conteniendo una estructura de directorios arbitraria de la cual se extraerán directamente en memorial los OIS requeridos, los cuales serán destruidos cuando ya no se usen, realizándose todo ello de forma transparente para el usuario. Este será el método por defecto en los módulos por ser el método más limpio y claro.

A esta compresión a la que todo el mundo se encuentra habituado y que se realiza a nivel de manejador de ficheros[9] se le llamará también archivado, empaquetado o macro-compresión, en contraste con la micro-compresión o compresión a nivel de serialización, que consiste en una serie de rutinas de bajo nivel para la escritura y lectura de cadenas de texto comprimidas de forma independiente y compatible con la indexación interna de los ficheros del OIS; para permitir así la posterior carga parcial de objetos sin necesidad de llevar a cabo una descompresión total del OIS. Esta se realiza mediante la librería open-source **BZip2**[10]. Para utilizarlo basta con pasarle a Ois.Store una ubicación sin extensión.

En proyectos grandes lo ideal es almacenarlos directamente en la base de datos en lo que llamaremos un **Ois.db**[11] en lugar de en ficheros para poder tener un método de almacenamiento masivo seguro y flexible con el que poder hacer búsquedas de forma sencilla y evitar los riesgos del acceso simultáneo y aumentar la seguridad en todos los sentidos sin tener que reinventar la rueda.

Una forma relativamente sencilla de programar sería almacenar en un campo blob en el que imbuir el directorio OIS completo de forma compacta e indisoluble. Cuando se requiera su lectura se bajaría el contenido a un directorio local y se usaría el OIS de la forma ordinaria.

El inconveniente sería la lentitud en el caso de procesos que requieran múltiples lecturas totales o parciales de un mismo OIS. Se haría necesario tener un sistema de control de cambios

Otra posibilidad es hacer otro diseño en el que cada archivo del OIS iría a un campo blob distinto y después programar las funciones de acceso directo de escritura y lectura parcial o total directamente en memoria sin pasar por copias locales.

[8] Ver http://www.artpol-software.com/ZipArchive
[9] aún sin implementar
[10] Ver http://www.bzip.org
[11] aún sin implementar

6.API de funciones y estructuras OIS

OIS implementa una serie de objetos que se pone a disposición del usuario para que pueda utilizar el sistema de la forma más versátil para extraer el máximo rendimiento. Sin embargo, muchas de las características de OIS no se utilizan en la mayoría de los casos por lo que sólo suponen una complicación para el usuario y es conveniente incluir funciones *ad-hoc* para cada caso.

Algunas cosas están implementadas en C++ y otras en el propio TOL. Las capacidades básicas genéricas de OIS están implementadas en C++ por razones de eficiencia y de accesibilidad a propiedades privadas de los objetos que no son visibles para el usuario de TOL.

Todas las estructuras, funciones y variables empiezan por el prefijo **Ois.** de forma que es sencillo encontraralas con el buscador de TOLBase junto con su descripción detallada.

Estas son las funciones más comunes en el uso simplificado de OIS:

- Para el manejo de módulos OIS precompilados se usará **Set Ois.UseModule(Text tolFile)**. El fichero TOL será compilado sólo si es necesario para reconstruir la imagen OIS, o sea, si no existe por ser la primera vez o haber sido destruido posteriormente, o bien si no está actualizado, o sea, si se modifica alguno de los ficheros fuente incluidos. La imagen OIS se generará la imagen OIS dentro del directorio especificado en la variable global Text Ois.DefRoot. Para usar un fichero TOL normal como un módulo OIS basta con llamar a Ois.UseModule en lugar de a Include.
- Para el manejo de imágenes OIS de caché de datos se usará la llamada compuesta

  Set Ois.UseModule(Ois.CheckUpdCaducity(tolFile,caducityDays)).

  Si la imagen OIS de un módulo correspondiente a un fichero .tol, no existe, o no está actualizada o ha transcurrido el plazo de caducidad especificado desde su creación entonces se destruye la imagen para que vuelva a ser reconstruida en la próxima carga.
- Para el uso de OIS como almacén estático de datos se dispone de la función **Real Ois.Store(Set data, Text path_)** que crea una imagen OIS estática de un conjunto arbitrario de objetos TOL sobre la que no se podrá hacer ningún chequeo de integridad referencial. Se puede especificar un camino usual relativo o absoluto o también un camino absoluto por defecto de OIS empezando por "ois:/" lo cual será reemplazadopor el contenido de la variable Text Ois.DefRoot
- Carga de imágenes con **Include**: La función Include es capaz de reconocer una imagen OIS compacta si se le pasa la ubicación de un fichero con extensión .oza o bien una imagen micro-comprimida si se le pasa un directorio sin extensión.