ACE-технология

Справочное руководство

Днепровский Проектный Институт
13 мая 1996

Оригинал-макет Справочного руководства по ACE-технологии выполнены Днепровским проектным институтом. С замечаниями и предложениями по поводу с ACE-технологии следует обращаться по адресу: Украина, 320008, Днепропетровск, ул. Театральная 6, Днепровский Проектный Институт, Комплекс Машинных Технологий, либо по E-mail: ace@dpi.dp.ua

Торговые знаки ДПИ

Следующие торговые знаки являются торговыми знаками Днепровского Проектного Института: ACE, ACE-технология, ACESOFT, SRTM, SRTM-среда, ARCH, ARCH-среда, TF, Табличные формы, FRM.

Торговые знаки других фирм

Следующие торговые знаки являются зарегистрированными торговыми знаками фирмы Autodesk, Inc: AutoCAD, AutoLISP, Autodesk Animator.

Следующие торговые знаки являются торговыми знаками фирмы Autodesk, Inc: ACAD, AME, ADS, AutoCAD SQL Extension, AutoCAD SQL Interface, DXF, FLI, 3D Studio, Автокад, Автолисп, Расширение по объемному конструированию, РОК, Система разработки приложений Автокада, СРП, SQL-интерфейс Автокада и Расширение Автокада по SQL-интерфейсу.

Все другие названия программ и оборудования являются торговыми знаками или зарегистрированными торговыми знаками соответствующих фирм.

Оглавление

Введение.

Истроический экскурс.

Рассмотрим процесс развития материального производства. Пока создавались достаточно не сложные объекты типа элементарной хаты, простейшие предметы быта или орудия производства- для их реализации достаточно было словесного описания "из уст в уста". Древний человек мог позволить себе создавать свое жилище “с чистого листа” пользуюясь только ранее приобретенными навыками. Создание мотыги или простейшего кувшина, также не требует какой-либо документации. По мере усложнения объектов производства и по мере усложнения производственных отношений появилась необходимость предварительной проработки объектов производства, а также необходимость передачи представлений об объекте другим членам общества. В самом деле, практически невозно представить себе чтобы на свет, безо всяческих схем, рисунков, текстов, могли бы появиться такие величесвенные создания человека как древнеегипетские пирамиды, храмы древней Греции, всевозможные машины и механизмы. Можно без преувеличения утверждать, что их появлением мы обязаны неизвестному гению, первым решившим изобразить объект производства в виде схемы или рисунка. По своей значимости это изобретение является ни чуть не менее важным, чем изобретение колеса. Появление технической документации на объекты материального производства было вызвано не стремлением конкурировать с методом "без техдокументации", а ПОЛНОЙ НЕВОЗМОЖНОСТЬЮ выполнить в натуре любой сложный объект, иначе, как предварительно разработав такую документацию.

Таким образом процесс эволюции процесса материального производства потребовал появления проекта (модели) объекта производства в силу следующих причин:

- усложнения обьекта, его полное представление во всех деталях одновременно стало более невозможным;
- в связи с разделением труда, стало необходимо давать разным исполнителям различную, но взаимоувязанную информацию;
- еще до начала работ появилась необходимость соотнести все необходимые ресурсы со своими возможностями и, при необходимости, откорректировать цель, (либо отказаться от нее вовсе - вспомним Робинзона, построившего лодку);
- стала желательной возможность проиграть варианты - что будет после реализации проекта;
- если обьект выполняется "на заказ", то заказчик при помощи проекта подтверждает свои ожидания.

По мере развития и усложнения объектов производства развивалась и техническая документация - от простейших схем и рисунков до современных чертежей. Нет сомнений в том, что в наши дни осуществляется ГЛОБАЛЬНЫЙ ПЕРЕХОД от бумажного описания объекта к определенным образом структурированной компьютерно-ориентированной информации. Компьтер не просто помогает заменить бумагу как носитель информации (хотя это и наиболее распростаненный в настоящее время подход), он позволяет создать качественно новую модель объекта. С уверенностью можно сказать, что последствия такого перехода будут не менее революционными для развития цивилизации, чем в свое время появление техдокументации.

Важно понять, что автоматизация инженерного труда, в результате которой другими машинными методами создавались те же бумажные документы - лишь этап, приближающий общество к ГЛОБАЛЬНОЙ ЦИРКУЛЯЦИИ компьютерно-ориентированной информации в процессе материального производства. Именно ради этой КОМПЛЕКСНОЙ ЦЕЛИ и стоит вкладывать средства и осуществлять государственную поддержку развития соответствующих отраслей. Именно эта комплексная цель обеспечит не просто эффективность, а СОХРАНЕНИЕ вообще соответствующих производств, (также немыслимых сейчас без традиционной бумажной техдокументации, как через какое-то время - без компьютерно-ориентированнной). Выбирать можно лишь между тактическими моментами, при решении вопросов, связанных с автоматизацией инженерного труда, стратегически выбора нет - так же как когда-то не было выбора при переходе на бумажные носители. Альтернатива - дикость.

Что даст оговоренная глобальная циркуляция и какие должны быть ее особенности? Оговоримся, мы имеем ввиду не просто машино-ориентированную информацию, которая может являться машинными форматами традиционной техдокументации (что в основном сейчас и имеет место). Информация должна быть организована таким образом, чтобы работать с ней можно было "в общем виде", она должна обслуживать "любые мыслимые" потребности, т.е. являться Унифицированной Моделью Объекта (УМО).

Работы, проводимые Днепровским Проектным Институтом (ДПИ) по "Отраслевой программе САПР капитального строительства" МИНМАШПРОМа привели авторов к следующим выводам :

1. Какую информацию должна содержать УМО.
   1.1. История "определения цели", т. е. после каких действий (процедур обработки), обращения к каким внешним знаниям, и кем утвержденного выбора мы вышли на данную УМО. для описания данного обьекта. Простое текстовое описание "определения цели". Какие ближайшие решения по " определению цели", были отвергнуты кем и по каким аргументам. Общественное мнение по

данному проекту. Этот информационный массив должен иметь большое значение для правильной кадровой политики - у каждого проекта должны быть герои, несущие всю полноту ответственности.

1.2. Данные определенным образом структурированные, описывающие обьект со всех необходимых понятийных сторон (в том числе и в натуре, с учетом замен, отклонений, реконструкций), совместно со ссылками на допускаемые процедуры обработки этих данных. Эту группу информации мы будем описывать особо и детально, сейчас же отметим важность обеспечения в описании органического триединства данных, процедур обработки, протокольной информации (либо соответствующих ссылок на данные, процедуры, протоколы). Этот раздел будет составлять замену традиционного понятия "проект" в виде чертежей (т.е. "электронная модель объекта").

1.3. Список полуфабрикатов, материалов участвующих в создании обьекта, либо ссылки на соответствующие внешние знания. История каким образом с обращением к каким внешним знаниям создана данная информация, когда, кто за нее отвечает.

1.4. Список процедур технологической обработки (обьекта в натуре) включающий:

- - описание технологических шагов по преобразованию обьекта в единстве со списками соответствующих полуфабрикатов и материалов.
- - в результате какого обращения и к каким внешним знаниям получен данный раздел и кто несет за него ответственность.

1.5. Информация, являющаяся результатом обращения к внешним знаниям по коньюктуре рынка, согласования с подрядчиком, удостоверяющяя наличие необходимых ресурсов:

- - для разработки детального проекта.
- - для начала работ по выполнению проекта в натуре.
- - Кто принял окончательное решение, что ресурсов достаточно.

1.6. Информация, являющаяся подтверждением фактических характеристик полуфабрикатов и материалов в натуре.

1.7. Информация, являющаяся подтверждением фактических характеристик машин, обеспечивающих технологические процедуры обработки обьекта. В результате каких процедур получены данные, кто несет за них ответственность.

1.8. Данные, обеспечивающие управляющими программами технологические машины. Процедуры обработки описания обьекта (по 2.) обеспечивающие удобную информацию для персонала технологических машин. Информация, являющаяся результатом работы этих машин (т.е. обратная связь).

1.9. Информация по анализу построенного обьекта, какими процедурами проверялось качество, какие проведены описания, кто несет ответственность за приемку. Какие необходимы доработки обьекта и по каким причинам, после обращения к каким внешним знаниям получены технологии доработок обьекта. Результаты этих доработок.

1.10. История получения обьекта от "определения цели" и до ввода вэксплуатацию и далее по внесению изменений и дополнений в обьект в процессе эксплуатации. Кто хотел бы внести изменения, какие, даты передачи данных подрядчику для проверки возможности изменений и их санкционирования, дата возврата описания, реализация изменений.

2. Что должна обеспечивать УМО:

2.1. Обеспечивать информационный обмен с внешними знаниями.

2.2. Обеспечивать циркуляцию всей УМО либо ее частей между соисполнителями работ.

2.3. Обеспечивать одновременный доступ различных соисполнителей.

2.4. Функцию описания истории, как побочный результат проектирующих и других процедур.

2.5. Реализовать проектирование с обособленными видами информации:

- собственно модель обьекта.
- процедуры обработки модели обьекта.
- списки элементов, составляющих модель обьекта.
- список полуфабрикатов и материалов требующихся для создания обьекта.
- список технологических процедур обработки по созданию обьекта в натуре.
- история создания обьекта.
- обеспечивать быструю и удобную интерпретацию данных с целью :
- представление обьекта во всех деталях с разбиением их на части.
- интерпретация цели объекта
- описание работы технологических машин
- многое другое

2.6. В процессе работы с УМО. должно быть возможно переключение языка общения с человеком. (русский, английский, украинский, казахский и т.д. без ограничения)

2.7. В процессе работы если, что-либо непонятно, должна одинаковым образом вызываться справка:

- звуковая
- статическая текстовая

- статический рисунок
- мультфильм
- голограмма
- и т.д.

2.8. При проектировании, после ввода порции информации, должно осуществляться контролирование ее на интервалы и логику с опорой на ранее введенные данные.

3. Что должна выполнять УМО в плане информационного обмена ?

3.1. Исследование обьекта (т. е. ответы на вопросы что будет, если ...)

3.2. Обеспечивать автоматическую передачу данных и процедур обработки УМО из внешних знаний в УМО в том числе :

- как результат работы управляемых технологических машин (п 3.6.2)
- как результат испытания физических моделей (п 3.6.3)
- как результат зондирования всего построенного объекта на предмет приемки

3.3. Структурное единство данных, процедур обработки, истории соответствующего материального производства.

3.4. Модификацию процедурами автоматизированной обработки, в том числе:

- "стандартными" процедурами автоматизированной обработки УМО (поступающими из внешних знаний)
- "индивидуальными" процедурами автоматизированной обработки (например, созданными для данного случая)

3.5. Модификацию "волевым методом" т.е. применение "ручного" проектирования средствами графических систем. (Всегда будут оставаться области приложений, для которых пока не созданы "стандартные" процедуры обработки). При этом соответствующая информация должна записываться в "историю" УМО.

3.6. Интерпретация многочисленными процедурами представления УМО во внешний мир (Заметим, что эта функция является конечной целью любой модели):

3.6.1. Просмотр заинтересованными лицами (проектировщики, строители, заказчики, контролеры ...) на всех стадиях шага материального производства. Тем самым реализуются потребности, связанные :

- с разделением труда;
- с наглядным представлением объекта;
- с получением разнообразных справок по объекту, в том числе по распределению юридической ответственности.

3.6.2. Выработка управляющей информации для управления технологическими машинами создания обьекта (в перспективе с выходом на нанотехнологии).

3.6.3. Выработка управляющей информации по созданию физических моделей обьекта.

3.7. Автоматическую установку в УМО ключей защиты, соответствующим образом реагирующих на перекачку УМО между потребителями.

3.8. Распределение юридической ответственности между участниками описываемого шага материального производства и поставщиками полуфабрикатов и технологических машин (одна из нагрузок "протокольной информации").

Фундамент будущей модели объекта.

Фундамент для создания УМО уже существует и достаточно широко используется (правда в более скромных целях) - это объектное (или объектно-ориентированное) программирование. Объекты, с самого начала, задумывались и разрабатывались, как средство для создания максимально приближенной модели реального мира. Благодаря этому объекты и объектное программирование изначально отвечают большинству требований к УМО. В настоящее время объектное программирование завоевало самую широкую популярность. Паскаль, Си++, Лисп, Smalltalk, Java и даже специальные расширения SQL - вот далеко не полный перечень языков, позволяющих сегодня использовать идеологию объектного программирования. Каждый из имеющихся инструментов имеет свои особенности реализации объектного программирования. В последнее время растут как грибы объектно-ориентированные базы данных, а среди пользователей и специалистов активно обсуждается вопрос какая из операционных систем имеет более объектный интерфейс. Объекты постепенно становятся неотъемлемой частью любого нового программного продукта. И это не просто дань моде - потенциальные возможности этой технологии настолько широки, что заставляют многих идти на довольно крутое изменение привычного стиля работы.

Несмотря на все несомненные успехи объектного программирования, необходимо отметить, что большинство систем построенных с использованием данной технологии все еще весьма далеки от требований УМО. Подавляющее число современных объектов СТАТИЧНЫ, большинство из них по-прежнему не может

обрести новые свойства (и/или методы) в процессе эксплуатации. Такое положение вещей не позволяет создать по настоящему открытую, развиваемую систему. Исходная структура большинства языков не позволяет естественным образом создать действительно динамический объект. Пожалуй только ЛИСП со своей списковой структурой может являться базовой динамичной средой для успешной реализации УМО. Простота добавления новых свойств в ассоциативные списки, возможность динамической генерации функций, функциональный подход - далеко не полный перечень тех преимуществ, которые может предоставить разработчику ЛИСП-среда. Следует отметить, что часть из перечисленных возможностей можно путем ухищрений реалиизовать и в традиционных языках программирования, но работа с данными на основе LISP-мышления позволяет это делать по ходу развития программного продукта проще и естественнее, проникаясь удивительным чуством признательности создателям LISP-культуры.

Для чего необходима ACE-технология?.

Анализ нынешнего положения дел в области САПР наглядно показывает на то противоречие, которое возникает между желанием иметь единую универсальную систему с общим интерфейсом, возможностью сквозной обработки данных различными подсистемами и высокой эффективностью узкоспециализированных задач. Единственным решением данной проблемы, на наш взгляд, является разработка базовых систем с открытой архитектурой и дополняемых различными специализируемыми модулями. За последнее время в программировании выделилось направление ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ (ООП), которое может послужить хорошей основой для разработки ОТКРЫТЫХ систем. Нашей целью являлась попытка реализации методов ООП в среде Автокада, и разработка с их помощью ОТКРЫТОЙ базовой системы САПР объектов капитального строительства. При этом, особое внимание уделялось именно открытости системы. Добавление новой программы, соответствующей спецификациям системы, в идеале, не должно вносить изменений в уже существующие. При этом, становится чрезвычайно важным та система соглашений, которая ложится в основу всей среды. Именно от этих соглашений зависит - насколько сложным будет процесс модификации системы. К нашему огромному сожалению в Автокаде нет готовых инструментов для ООП, реализация же методов ООП, в полной динамике, возможна только на ЛИСПе. И хотя Автокад и Автолисп не являются, по нашему мнению, идеальной платформой для реализации задуманных концепций - все же это единственный вариант, позволяющий получить работающий прототип системы за приемлемый отрезок времени. Огромное влияние на систему оказали более ранние разработки - SRTM и ARCH, на которых были опробованы некоторые из методов и концепций. Среди таких концепций, получивших свое дальнейшее развитие, такие основополагающие механизмы, как вычисляемые слоты и наследование. Оба эти механизма претерпели существенные изменения, значительно расширяющие область их применения. Добавлен ряд новых концепций, которые отсутствовали в предыдущих разработках - среди них “разворачиваемые” ключи, методы-демоны и некоторые другие. К большому сожалению некоторые концепции не смогли быть включены в настоящую работу ввиду большой трудоемкости их реализации, так, в настоящее время остались не реализованными такие важные моменты, как история развития объекта, разграничение прав доступа. Кроме того, по-прежнему больным вопросом остается проблема разработки и наполнения необходимых баз. Однако, несмотря на все эти замечания, уже сейчас можно смело утверждать, что сделан огромный шаг в верном направлении.

Соглашения по шрифтам.

В настоящем руководстве приняты следующие соглашения по шрифтам:

HelvDL курсив	Имена каталогов (директорий), имена и типы файлов, имена фреймов: *ace_root.fr, /ace/a-main/fun, .slb*
Моноширинный	Переменные, типы данных, значения и примеры программ Автолиспа и Си: pt1, NIL, pi, INT
Моноширинный полужирный шрифт	Имена функций Автолиспа и Си: **command, xform**
Моноширинный курсив	Формальные аргументы, заданные в определениях функций: **(@Get-Slots** *slots frame opt* **)** Функция **@Get-Slots** требует на вход три обязательных аргумента *slots frame opt.*
HELVDL	Имена слотов (имена ключей ассоциативных списков):
ВСЕ ПРОПИСНЫЕ	NM, UP, FML

Обозначения для описания синтаксиса.

Для описания синтаксиса используется расширенный формализм Бэкуса-Наура (РБНФ). Синтаксические понятия (нетерминальные сисмволы) изображаются русскими словами заключенными в угловые скобки, выражающими их интуитивный смысл. Символы яхыка (терминальные символы) - это строки заключенные в кавычки или слова написанные курсивом. Каждое синтаксическое правило (продукция) имеет вид

S=E.

где S - синтаксическое понятия, а E - синтаксическое выражение, изображающее множество сентенциальных форм (последовательностей символов), которые обозначают S. Выражение E имеет вид

T1 | T2 |...|Tn (n>0),

где Ti является термом выражения E. Каждое Ti обозначает некоторое множество сентенциальных форм, а E изображает их объединение. Каждый терм T имеет вид

F1 F2 ... Fn (n>0),

где Fi - множитель терма T. Каждое Fi обозначает некоторое множество сентенциальных форм, а T изображает их конкатенацию. Конкатенация двух множеств последовательностей символов - это множество последовательностей, состоящее извсех таких конкатенаций, когда последовательность из первого множителя продолжается последовательностью из второго множителя. Каждый множитель F - это или символ (терминальный или нетерминальный), или он имеет вид [E] и изображает объединение множества E и пустой последовательности или {E} и изображает объединение пустой последовательности и любой из последовательностей E, EE, EEE, ... Круглые скобки можно использовать для группировки термов и множителей.

РБНФ может описать свой собственный синтаксис, например, следующим образом:

<синтаксис> ::= {<правило>Ї}.
<правило> ::= <НТСимв> “::=“ <выражение> “.”.
<выражение> ::= <терм> {“|” <терм> }.
<терм> ::= <множитель> { <множитель> }.
<множитель> ::= <Тсимв> | <НТСимв> | “(“ <выражение> “)” | “[“ <выражение>“]” |
“{“ выражение “}”.
<Тсимв> ::= ”\”” <имя символа> “\“”.
<НТСимв> ::= “<“ <имя символа>“>“.
<имя символа> ::= *строка*.

Для обозначения символа “ (двойной кавычки) в РБНФ будем использовать комбинацию символов \”.

Основные понятия ACE.

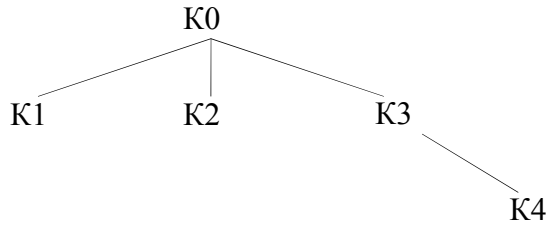
Введение в объектное программирование.

ACE-технология построена на представлении ОБЪЕКТОВ в виде ЛИСП-списков и примитивов Автокада. Каждый объект в системе представлен в виде некоего ассоциативного списка - будем называть его ФРЕЙМОМ. Каждый фрейм может содержать произвольную информацию, характеризующую этот фрейм. Эта информация представляется в виде ассоциативных пар ключ-значение, которые мы будем называть СЛОТОМ. Кроме слотов представляющих СВОЙСТВА объекта, существуют также и служебные слоты. Система поддерживает несколько типов слотов, отличающихся своим форматом.

Строго говоря, можно выделить два вида фреймов в системе - ОБЪЕКТЫ и КЛАССЫ. Строгую границу между этими понятиями провести не всегда просто, однако можно сказать, что КЛАССЫ представляют собой описание некой общей информации какой-либо группы ОБЪЕКТОВ, а ОБЪЕКТЫ - описание какого-либо конкретного представителя того или иного КЛАССА. Будем говорить, что объект А является ЭКЗЕМПЛЯРОМ некоторого класса К, если А объект класса К. Один ОБЪЕКТ может выступать в качестве ЭКЗЕМПЛЯРА нескольких КЛАССОВ одновременно. Общие свойства нескольких различных классов могут быть представленны в виде отдельного класса. Такой специальный класс будем называть НАДКЛАССОМ соответсвующих классов, а сами эти классы - ПОДКЛАССАМИ этого надкласса.

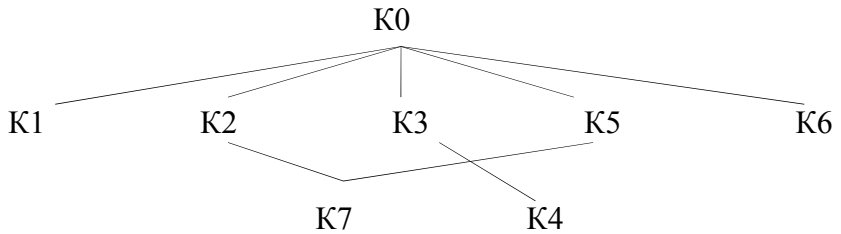
Рассмотрим эти понятия на маленьком примере. Пусть есть описания нескольких классов треугольников: класс ПРЯМОУГОЛЬНЫХ треугольников (К1), класс ТУПОУГОЛЬНЫХ треугольников (К2), класс ОСТРОУГОЛЬНЫХ треугольников (К3). Если мы захотим объеденить все треугольники в один класс, то может ввести класс ПРОИЗВОЛЬНЫХ треугольников (К0) и сделать все уже существующие классы треугольников (К1, К2, К3) его субклассами. Если после всего этого нам понадобится класс РАВНОСТОРОННИХ треугольников (К4), то мы сможем описать его как субкласс класса ОСТРОУГОЛЬНЫХ треугольников. В результате мы получаем некоторую схему иерархий классов:
В данной иерархии класс К3 является ПОДКЛАССОМ класса К0 и одновеменно НАДКЛАССОМ класса К4.

рис. А



При этом класс К4 наследует как свойства класса К3, так и свойства класса К0 (но не классов К1 и К2). В более общем случае каждый класс может иметь несколько надклассов - продолжим рассмотрение нашего примера, допустим, что мы захотели добавить классы РАВНОБЕДРЕННЫХ (К5) и РАЗНОСТОРОННИХ (К6) треугольников. Поскольку и равнобедренным и разносторонним с одинаковым успехом может быть ОСТРО-, ТУПО- и ПРЯМОУГОЛЬНЫЙ треугольники, то достаточно очевидно, что эти два класса должны быть независимы от К1, К2 и К3. Таким образом мы приходим с схеме изображенной на рис. В. Необходимо обратить внимание на класс К7, который унаследовал от класса К2 один тупой угол, а от класса К5 - два равнных ребра, таким образом класс К7 является, по сути, описанием класса РАВНОБЕДРЕННЫХ

рис. В

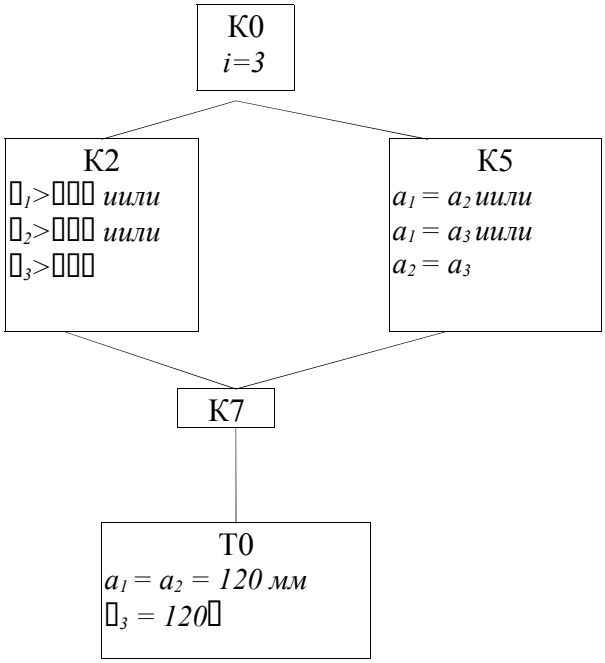


ТУПОУГОЛЬНЫХ треугольников, при этом собственно описание состоит только из ссылок на надклассы К2 и К5.

Все данные какого-либо объекта или класса условно можно разделить на описательные, или СВОЙСТВА, и процедурные, или МЕТОДЫ. Оба типа данных вместе и образуют неразрывное целое - объект либо класс (в дальнейшем, для краткости, будем говорить только об объекте, подразумевая что это в той-же мере касается и класса, если специально не оговорено противное). СВОЙСТВА определяют все характеристики объекта, часть свойств объекта могут быть определены непосредственно в самом объекте, а часть в классах, экземпляром которых данный объект является. Рассмотрим пример на рис. С. Оставим из предыдущего

примера только относящиеся к делу классы, а также добавим свойства и опишем треугольник T0. В классе K0

рис. C



(общий класс треугольников) мы указали что число сторон (и/или углов) для всех треугольников равно 3. В классе K2 мы оговорили, что один из трех углов (и только один) больше прямого. А в классе K5 мы оговорили, что две из трех сторон (и только две) имеют равную длину. В классе K7, как мы уже говорили, никаких специфических свойств не задано, однако, в общем случае, это могла быть любая информация - например цвет. И хотя класс K7 имеет довольно много свойств описывающих равнобедренный тупоугольный треугольник ее все еще недостаточно, чтобы можно было, например, построить его. Все перечисленные свойства носят общий или РОДОВОЙ характер. Одних этих свойств не достаточно, чтобы говорить о конкретном треугольнике. Это необходимо иметь в виду, когда мы хотим создать экземпляр равнобедренного тупоугольного треугольника. Мы должны каким-то образом дополнить РОДОВЫЕ класса свойства таким образом, чтобы однозначно определить конкретный треугольник. В нашем примере это сделано при помощи задания угла и двух прилегающих сторон, хотя существуют и другие очевидные способы. Свойства, которые описывают только конкретный экземпляр будем называть ИНДИВИДУАЛЬНЫМИ. Зачастую бывает сложно, а порой и невозможно провести четкую границу между родовыми и индивидуальными свойствами, поскольку в различных ситуациях одни и теже свойства могут выступать и в роли индивидуальных и в роли родовых - в конечном счете все зависит от конретной рассматриваемой ситуации и степени приближения к действительности. Например, до тех пор пока у нас есть единственный треугольник T0, то можно говорить, что его свойства индивидуальны, однако если нам понадобиться несколько аналогичных треугольников с заданными цветом и привязками в пространстве, то описание экземпляра T0 вполне логично преобазовать в описание соответсвующего класса, для которого величина угла и размеры сторон из индивидуальных свойств превратятся в родовые. Зачастую одни задачи могут рассматривать один и тот-же фрейм как объект, в то время как в других он может выступать в роли класса. В конечном итоге, все зависит, по-видимому, от степени абстрагированности той-или иной задачи.

Чаще всего родовые и индивидуальные свойства просто дополняют друг друга, однако возможны случаи когда по тем или иным соображениям индивидуальные свойства могут переопределять родовые. К данному случаю необходимо подходить очень внимательно. Существует целый ряд потенциальных источников ошибок связанных с этой ситуацией. Рассмотрим одну из них. Допустим, что мы превратили T0 в описание класса и для ускорения работы системы решили сразу же определить и записать остальные параметры (два остальных угла, третья сторона, периметр, площадь) треугольников этого класса, а не соответсвующие им выражения. После этого в одном из экземпляров мы заменили величину угла индивидуальной. Совершенно очевидно, что после записи в класс, в качестве родовых свойств, единожды вычисленных значений, мы прийдем к противоречию между индивидуальным значением угла и родовыми значениями остальных параметров - мы получим описание невозможного набора свойств. В сложившейся ситуации, мы будем вынуждены либо отказаться от замены внутри класса выражений вычисленными значениями, либо на уровне конкретного экземпляра, одновременно с замещением угла, заместить и другие связанные с ним свойства. Этот пример наглядно демонстрирует насколько резко может изменяться ситуация с течением времени. Именно из-за этого всегда, по мере возможности, следует выбирать наиболее общие способы моделирования. Совершенно очевидно, что в приведенном выше примере ценой довольно сомнительного повышения эффективности, мы получили гораздо более неустойчивую систему, и поэтому следует признать идею замены выражений вычисленными значениями не вполне оправданной (в данном случае, если же вычисления настолько сложны что занимают ощутимый

период времени, то подобная замена и соответствующий механизм контроля могут оказаться вполне оправданными).

Методы аналогичны обычным функциям, они определяют как может быть использован объект и какие операции над объектом мы можем произвести.

Реализации объектов в ACE-технологии.

Каждый фрейм обязан содержать служебный слот NM, причем, для повышения быстродействия системы принято соглашение, что этот слот должен быть всегда первым слотом во фрейме. Этот слот представляет собой ВНУТРЕННЕЕ имя фрейма в системе. Это имя должно быть уникально в пределах системы. По этому имени осуществляется поиск фрейма, проверка подлинности фрейма, оно используется для отображения в режиме монитора с сокращенными ключами.

Слоты могут представлять не только свойства фреймов, но и МЕТОДЫ, определенные для данных объектов. В данный момент система поддерживает методы двух видов: “ВЫЧИСЛЯЕМЫЕ” слоты и “ДЕМОНЫ”. Условно их можно представить в виде “пассивных” и “активных” методов. Так, для определения значения “вычисляемого” слота необходимо просто указать его в списке читаемых слотов соответствующей функции. Пользователь может даже не подозревать, что тот или иной слот является “вычисляемым” - для него он ни чем не отличается от обычного. Хотя при определении одного “вычисляемого” ключа, может потребоваться определение ряда других “вычисляемых” и/или обычных ключей, тем не менее, это, обычно, не приводит ни к каким изменениям во фрейме. “Демоны”, в отличие от “вычисляемых” слотов, призваны как раз модифицировать определенным образом фреймы и реагировать на их изменения. “Демоны” запускаются автоматически при добавлении или модификации слотов при помощи специальной функции посылки СООБЩЕНИЙ. Каждый “демон” следит за определенным набором слотов от значения которых он зависит. Изменение какого-либо слота из этого набора приводит к активизации “демона”. Поскольку различные “демоны” могут иметь пересекающиеся наборы активирующих их слотов, то при изменении какого-либо слота может запускаться произвольное число “демонов”. Кроме того, результатом работы “демона” может стать модификация какого-либо слота (или группы слотов), что, в свою очередь, может привести к вызову других “демонов”. Таким образом, “демоны” можно охарактеризовать как “активные” методы, а “вычисляемые” выражения - как “пассивные”. Необходимо подчеркнуть, что для активизации “демонов” необходимо, чтобы все вносимые изменения выполнялись только функцией посылки “сообщений”.

Фрейм может содержать ссылки на “родительскую” информацию либо, говоря иными словами, на фреймы более “высокого” уровня. Необходимо отметить, что термин “высокого” относится, скорее, к уровню абстракции того или иного фрейма. Будем называть такие ссылки ССЫЛКАМИ ВВЕРХ. В общем случае число “родителей” не ограничено. Если какой-либо фрейм А содержит ссылку наследования на фрейм Б, то будем говорить, что фрейм А наследует свойства и методы фрейма Б. Таким образом, те свойства и методы, которые не определены во фрейме А, но описаны во фрейме Б, становятся НАСЛЕДУЕМЫМИ свойствами и методами фрейма А. В отличие от наследуемых слотов, будем называть слоты, определенные непосредственно в данном фрейме - ЛОКАЛЬНЫМИ слотами. Во время работы с фреймом наследуемые слоты доступны в такой же мере, как и локальные. Если некоторый слот определен и во фрейме А, и во фрейме Б, то локальный слот переопределяет наследуемый. Существует однако специальный тип слотов, который допускает не замещение, а слияние слотов, делая таким образом доступными слоты различных уровней. Такие слоты будем называть “СЛИВАЕМЫМИ”.

Кроме ссылок вверх некоторые фреймы могут содержать также и ССЫЛКИ ВНИЗ. Обычно такие фреймы представляют собой описание КЛАССОВ объектов, а ссылки вниз - список объектов, входящих в данный класс. Объект, входящий в состав класса, будем называть ЭКЗЕМПЛЯРОМ этого класса. Один объект может одновременно являться экземпляром нескольких классов. Строго говоря, такие прямые ссылки вниз не является безусловно необходимыми для построения функционирующей системы, однако их использование позволяет существенно упростить процесс просмотра экземпляров соответствующего класса. Ссылки вниз не влекут за собой наследования, а служат только для быстрого поиска экземпляров класса.

В системе всегда существует, так называемый, корневой фрейм. Корневой фрейм, является самым верхним фреймом в системе. Корневой фрейм содержит как наиболее общие свойства, так и наиболее общие методы. Каждый фрейм через промежуточные уровни может, в конце концов, сослаться на свойства или методы корневого фрейма, если они не были переопределены в одном из промежуточных фреймов. Цепочка наследования, однако, не обязана всегда подниматься до корневого фрейма. Некоторые ветви наследования могут заканчиваться на каком-либо из промежуточных фреймов. Из-за использования механизма множественного наследования некоторые фреймы могут проходиться несколько раз по различным путям. В настоящее время в системе не существует механизма осуществляющего соответствующий контроль. Эта особенность не оказывает никакого влияния на значение обычных слотов, однако может существенно изменять результат “сливаемых”.

Механизм наследования, в настоящее время, использует два механизма - механизм ПРЯМЫХ ССЫЛОК, т.е. непосредственного указания имени (внешнего имени ЛИСП-переменной) фрейма на который необходимо сослаться и механизм ССЫЛКИ ПО ПОИСКУ - этот метод обычно используется в тех случаях, когда: либо у фрейма нет внешнего имени, либо оно точно не известно, либо в тех случаях, когда оно может быть изменено в

дальнейшем. Оба механизма имеют свои преимущества и свои недостатки. Прямые ссылки существенно быстрее, чем ссылки по поиску, однако последние, как уже отмечалось, не требуют менее точной информации о искомом фрейме, что делает их более гибкими и менее чувствительным к вносимым в систему изменениям.

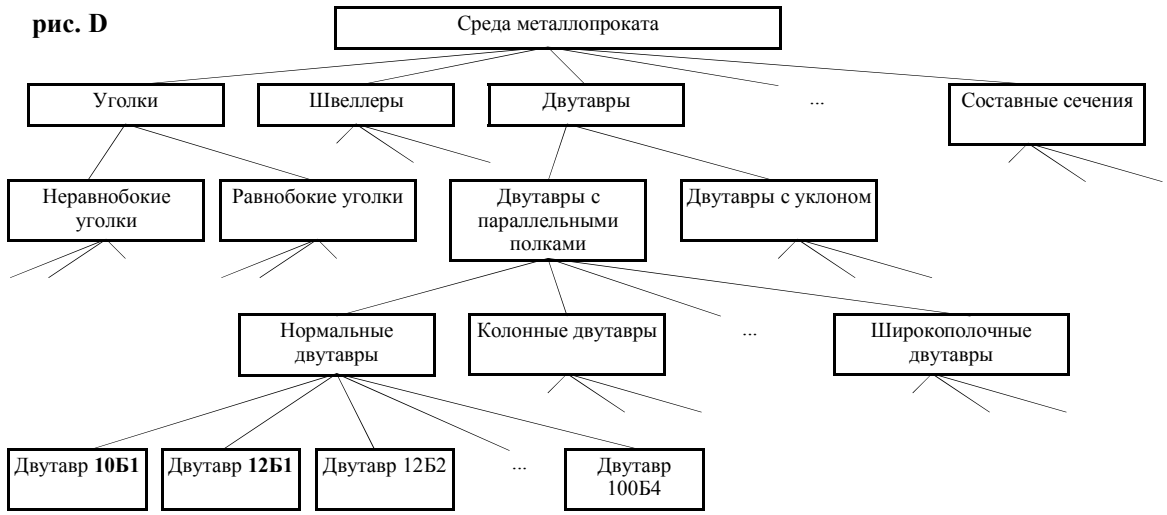
Структура ACE.

Структура ACE-сети.

Структура ACE-сети задает систему иерархий среди фреймов. Структура этой сети играет огромное значение для функционирования системы. Поскольку в системе широко используется механизм наследования, то правильное задание иерархических связей играет очень важную роль. Чрезвычайно важно при разработке системы выработать систему иерархических понятий и правильно ее представить в виде сети фреймов. При выработке системы понятий желательно исходить из конкретной прикладной области. Старайтесь найти и выделить общие свойства и методы у различных объектов и понятий предметной области. Нахождение хотя бы одного общего свойства у некоторой группы объектов может служить достаточным основанием для создания некоторого класса объектов. Один объект может одновременно являться экземпляром нескольких классов. Чем ближе будет Ваша классификация к реальной предметной области, тем точнее окажется модель и тем естественнее будут решаться связанные с ней задачи. Чем выше удастся Вам поместить, то или иное свойство, тем для большего круга объектов оно станет доступно, и тем легче его будет использовать и модифицировать. Необходимо помнить, что при необходимости можно всегда переопределить то или иное общее свойство, указав в нужном фрейме более конкретное, и поэтому возможно указывать свойства даже для тех групп понятий, у которых они не одинаковы для всех элементов этой группы.

Представим, что Вам захотелось создать базу описывающую стальные прокатные профили. Все многообразие прокатных профилей можно разбить на несколько групп - двутавры, швеллеры, уголки, тавры, сечения прямоугольного и круглого профиля. Каждую из этих групп можно разделить на подгруппы, например, двутавры - обычные, широкополочные, балочные и т.д., уголки - равнобокие и неравнобокие, круглые профили - трубы и сплошной круг и т.п. Продолжая деление подобным образом, в конце концов, мы добираемся до групп, состоящих из конкретных профилей - двутавр 16, двутавр 18 и т.д., при этом, число промежуточных уровней для различных профилей различно и не играет ни какой роли с точки зрения пользователя. Изобразим одну из возможных систем иерархий стального проката в виде схемы, приведенной на рис. D:

рис. D

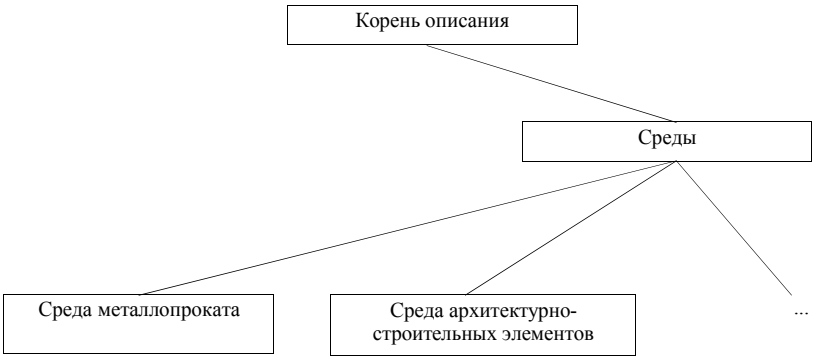


С одной стороны создание подходящей системы иерархий может являться довольно непростой проблемой, в особенности в тех случаях, когда рассматриваемая область достаточно сложна. Во многих случаях иерархия системы далеко не столь очевидна, как в рассматриваемом примере. Если же принять во внимание тот факт, что зачастую довольно сложно изолировать какую-либо систему иерархии от остального мира, то становится очевидно, что решение этой задачи требует определенных усилий и навыков. Однако, несмотря на все сложности ее создания, выделенная система иерархий способна принести выгоды с лихвой, компенсирующие все соответствующие затраты. Перечислим некоторые из них:

- значительное сокращение общего объема информации по сравнению с реляционным представлением информации за счет разнесения ее по уровням иерархии и использовании механизма наследования;
- выделение общей информации, простота ее модификации, создание общих (“родовых”) процедур;
- простота переопределения “родовой” информации и “родовых” процедур;
- сравнительная простота модификации иерархической схемы.

ACE-сеть может включать в себя множество подобных сред. Вполне естественно, что это влечет за собой появление дополнительных верхних уровней иерархии (см рис. E). Каждая такая среда представляет собой аналог отдельной базы данных (база данных сортамента металлопроката, база данных архитектурно-строительных элементов и т.д.) и может быть полностью независима от остальных. Конкретный набор сред зависит от типов решаемых задач и не является чем-то жестко зафиксированным, он может быть достаточно легко изменен.

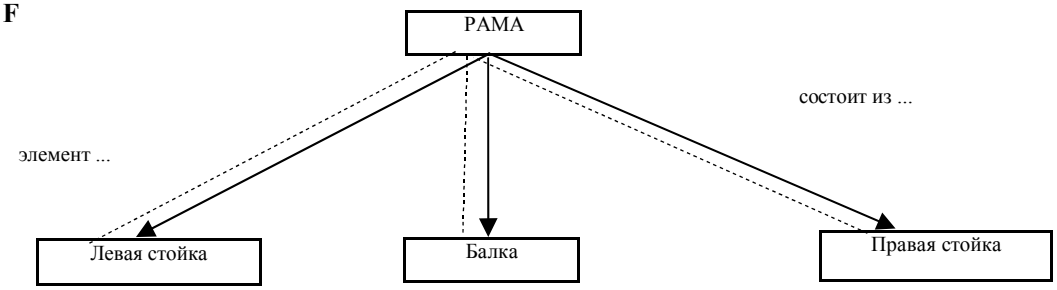
рис. E



Проектируемый объект, однако, состоит не из разнообразных баз данных, а из конкретных элементов. Каждый из этих, составляющих объект в целом, элементов сам, в свою очередь, может являться составным. Таким образом, параллельно с деревом иерархий баз данных возникает дерево иерархии проектируемого объекта. Это дерево, строго говоря, представляет собой некую модель проектируемого объекта, которая и является основной целью процесса проектирования в целом. Рассмотрим в виде примера простейшую "П"-образную раму, состоящую из двух стоек и балки.

Основным отличием системы иерархии модели объекта от баз данных, как легко заметить, является наличие двунаправленных связей типа "состоит из..." и "элемент ...". Связь типа "элемент ..." имеет прямой аналог в системе иерархий баз данных, в то время как связь типа "состоит из ..." является специфичной именно для модели объекта.

рис. F



Однако представленную схему, приведенную на рис. F, нельзя считать полной, так как на ней представлен только один из возможных информационных срезов. Для проведения прочностных расчетов, например, нам также понадобится информация о узлах и нагрузках, для каких-либо других целей может потребоваться иная произвольная информация, которую трудно представить в данном информационном срезе. Поэтому для одного объекта может быть построено несколько независимых систем иерархий.

Структура фреймов.

Как уже отмечалось, фрейм представляет собой довольно гибкую форму организации данных и функций, в этой связи довольно тяжело говорить о какой-либо структуре фрейма. Однако все же можно выделить некоторые общие моменты для всех фреймов.

"Внутренние" и "внешние" имена фреймов.

Каждый фрейм представляет собой некоторый ассоциативный список. Содержимое каждого фрейма в большой степени довольно произвольно. Существует, однако, ряд слотов обязательных для всех (или большинства) фреймов. Первым среди таких слотов необходимо выделить "внутреннее" имя фрейма. Формат этого слота предельно прост: *(NM . "ДВТБ_12Б2").* Этот слот используется для идентификации фрейма. По принятым в настоящее время (для ускорения этого процесса) соглашениям, слот 'NM всегда должен быть первым слотом в любом фрейме. Внутреннее имя должно быть уникально по крайней мере в пределах одной среды. Хотя появление двух различных фреймов не обязательно ведет к конфликтам, однако это потенциальный источник ошибок, которого следует всячески избегать. В настоящее время, к сожалению, отсутствуют средства

контроля за появлением “двойников” и вся ответственность ложится на разработчика. Поэтому необходимо тщательно планировать структуру среды во избежание подобных случаев.

Большинство фреймов кроме “внутреннего” имени имеют также и “внешнее”. Под “внешним” именем мы будем понимать имя символа Автолиспа, значением которого является данный фрейм. “Внешнее” имя, в отличии от “внутреннего”, не является обязательным атрибутом фрейма, так элементы семейств могут не иметь собственного “внешнего” имени. Внешнее имя фрейма, если оно присутствует, находится в слоте с именем ‘EXT_NM. Данный слот имеет следующий формат: (EXT_NM . символ). Этот слот является автоматически добавляемым и не требует своего явного указания в момент создания фрейма. Функции работы с фреймами сами добавляют этот слот при первом обращении к фрейму. Одинаковые “внешние” имена, строго говоря, могут иметь несколько различных фреймов даже в пределах одной среды. Однако эти фреймы должны быть определены в различных файлах. Если возникает необходимость поочередно прочесть два разных фрейма с одним “внешним” именем, то система вынуждена будет перегрузить фрейм перед вторым чтением, затерев при этом первый фрейм. Если ситуация требует постоянного чередования фреймов, то временные расходы на загрузку могут стать неприемлемо большими. В этом случае необходимо подумать о присвоении фреймам различных имен. Однако использование одних и тех же “внешних” имен может существенно снизить потребность в оперативной памяти доступной системе. Поэтому окончательное решение использовать одинаковые или разные “внешние” имена фреймов зависит от конкретной ситуации. Существует один аргумент в пользу использования различных имен, хотя он основан скорее на догадке, чем на точных сведениях: очень высока вероятность того, что Автокад работающий в защищенном режиме при недостатке оперативной памяти сбрасывает на диск и ЛИСПовские страницы. Если это предположение верно, то выгоднее использовать различные имена фреймов, не опасаясь нехватки памяти Автолиспа, экономя при этом довольно существенное время, так как загрузка обменных страниц должна выполняться существенно быстрее, чем загрузка файлов, поскольку не требует анализа файла и его преобразования во внутренний формат.

Указатели наследования.

Большинство фреймов содержат по крайней мере одну ссылку на родительскую информацию. Для задания таких ссылок используется специальный слот указания наследования. Этот слот имеет более сложный формат, чем слот “внутреннего” имени. Имя этого слота не является фиксированным, в зависимости от необходимости может быть использовано одно из возможных имен зарезервированных в системе. Если список зарезервированных для этой цели имен Вам представляется не полным, Вы можете его расширить, изменив соответствующий слот в корневом фрейме (слот ACE_UP). Примем для простоты, что имя этого слота ‘UP. Существует два вида задания этого слота - прямое указание фрейма и путем использования поиска. Рассмотрим вначале вариант с непосредственным указанием фрейма:

```
(UP
    (NM . <внутреннее имя фрейма>)
    (FR . <внешнее имя фрейма> <имя файла>)
    (PT . <путь к файлу>)
    ...
)
```

Как видно, для его задания необходимо знать как “внутреннее” так и “внешнее” имена фрейма. Кроме того, если фрейм еще не загружен или указанное “внутреннее” имя не совпадает с “внутренним” именем фрейма, то может понадобиться также информация и о файле(в том числе и о пути к нему), в котором описан искомый фрейм. Все это накладывает некоторые ограничения на возможность применения данного метода указания наследования. Определяющим в данном случае является “внешнее” имя фрейма, а “внутреннее” используется для подтверждения подлинности фрейма, таким образом для успешной ссылки должны совпасть оба этих параметра. Использование контроля подлинности фрейма необходимо, для обеспечения системы использующей не уникальные “внешние” имена.

Метод с использованием поиска требует задания несколько иной информации о фрейме:

```
(UP
    (NM . <внутреннее имя фрейма>)
    (FR  FIND  <имя среды>)
     ...
)
```

Для поиска необходимо указать только “внутреннее” имя фрейма и имя среды, в которой будет осуществляться поиск этого фрейма. Явное указание среды позволяет иметь одинаковые “внутренние” имена фреймов в различных средах, что делает систему более гибкой и менее требовательной с точки зрения согласования различных сред между собой, что, в свою очередь существенно облегчает процесс добавления новых сред к функционирующей системе. При добавлении новой среды нет необходимости подробно изучать уже имеющиеся и следить за уникальностью “внутренних” имен фреймов. Для использования данного метода требуется гораздо менее точная информация о фрейме, что может играть существенную роль в ряде случаев, кроме того такая ссылка менее чувствительна к вносимым в среду изменениям, так если по каким-либо причинам нам необходимо изменить “внешние” имена фреймов, то вариант с непосредственным указанием потребует внесения соответственных изменений во все фреймы, в то время как вариант с использованием

поиска останется без изменения. Однако у данного метода есть и существенный недостаток - гораздо более низкое быстродействие, вызванное большим временем поиска нужного фрейма.

Семейства и "безымянные" фреймы.

Фреймы могут содержать внутри себя СЕМЕЙСТВА других фреймов. Этот механизм необходим для реализации поиска фреймов, а также для движения по ACE-сети в режиме монитора. Для описания семейства используется специальный слот с именем FML. Формат данного слота довольно гибок и позволяет описывать разнообразные ситуации. Семейство представляет собой список составляющих его фреймов либо ссылок на них, в начало которого добавлен специальный "нулевой" элемент. Специальный элемент служит для сокращения расхода памяти и используется для задания общей информации, например указатели наследования и прочее. При обработке происходит объединение специального элемента с конкретным элементом списка, а затем лишь обработка полученного фрейма. При этом информация из "нулевого" фрейма всегда помещается после информации конкретного элемента, для того чтобы иметь возможность переопределить любые заданные в "нулевом" фрейме слоты. Проиллюстрируем это примером, для чего вернемся к примеру, приведенному на рис. D и продемострируем часть фрейма "Нормальные двутавры":

```
(FML
    ((UP ((NM . "Нормальные двутавры")
          (FR DWTB "DWTB.FR")
          ...
     )
     (BEFORE <слот предобработки>)
     (CUT <слот усечения>)
      ...
    )
    ((NM . "ДВТБ_10Б1") ...)
    ((NM . "ДВТБ_12Б1") ...)
    ((NM . "ДВТБ_12Б2") ...)
    ...
    ((NM . "ДВТБ_100Б4") ...)
  )
```

Файлы и фреймы.

На период между рабочими сеансами фреймы хранятся в файлах. Не существует никаких особых правил по размещению фреймов в файлах. Вы можете выделить для каждого фрейма свой отдельный файл, а можете собрать их все в один - существует лишь одно ограничение: нельзя записать в один файл два фрейма с одинаковыми "внешними" именами, поскольку при загрузке более позднее определение затрет более раннее. Чем меньше размер файла, тем быстрее выполняется его загрузка. Однако большее кол-во мелких файлов приводит к нерациональному расходу дискового пространства. Рекомендуется объединять в один файл, по крайней мере, такие взаимосвязанные фреймы, что при работе с одним, обязательно потребуется и другой, таким образом минимизируется и число файлов, а также число и время загрузок.

Слоты.

Все слоты в системе можно классифицировать по нескольким признакам- информационные и системные, сливаемые и наследуемые, простые и разворачиваемые, вычисляемые, демоны. Несмотря на все разнообразие форматов все эти слоты подчиняются ряду общих правил:

- каждый слот представляет собой ассоциативную пару ключ-значение;
- в качестве ключа всегда выступает некоторый ЛИСП-символ;
- значением может быть число (целое или вещественное), строка, символ, имя примитива, набор, ЛИСП-список - т.е. любой допустимый тип Автолиспе;
- независимо от типа значения пара строится с помощью функции **CONS**;
- в различных фреймах слоты с одним и тем же именем могут иметь различный формат.

Некоторые слоты могут иметь достаточно сложный формат, и являться фактически, фреймами во фрейме. В большинстве случаев это относится к системным слотам. Такие слоты используются непосредственно системой и обрабатываются особым образом. При разработке и/или наполнении системы разработчик должен отчетливо представлять себе формат каждого системного слота и специфические механизмы обработки. Системные слоты, в отличие от информационных, зачастую имеют строго зафиксированный формат.

Обычные слоты.

В системе под обычным слотом понимается слот, значение которого представлено непосредственно в том виде, в каком его возвращает функция чтения, не выполняя при этом никаких дополнительных операций, связанных с вычислением, сливанием и прочим. При отсутствии обычного слота в указанном фрейме функция

чтения слотов ищет первое вхождение слота по сети наследований, т.е. имеет место обычный механизм наследования. Обычные слоты нет необходимости объявлять каким-либо особым образом. Все слоты, по-умолчанию, изначально принимаются как обычные. Для объявления слота, как специального используются особые методы объявления, которые будут рассмотрены в соответствующих разделах.

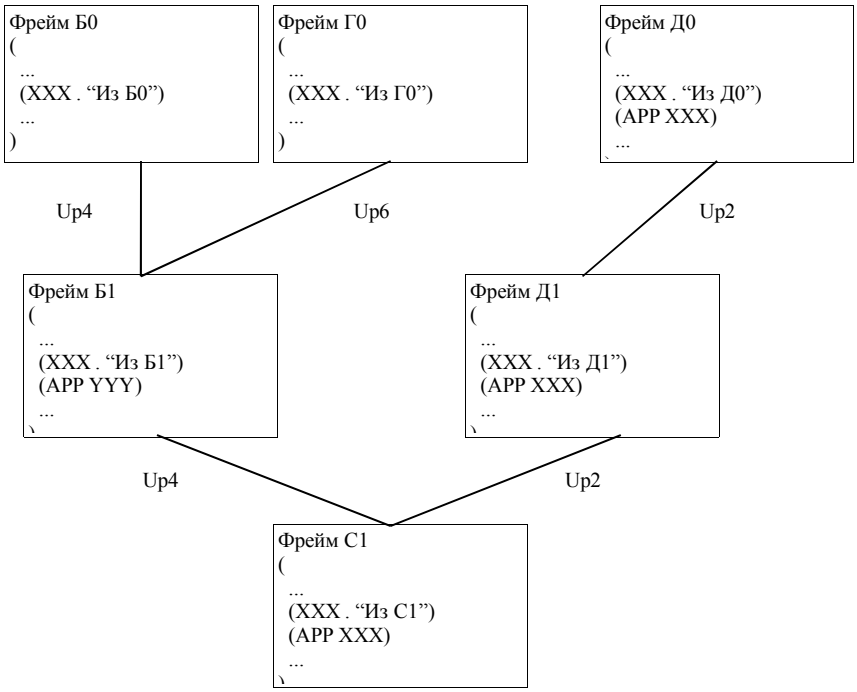
Примеры обычных слотов:

(H . 220) - значение 220 - атом
(B 220) - значение (220) - список из одного элемента
(TU “ГОСТ 2222.88”) - значение (“ГОСТ 2222.88”) - список
(PT0 . (10 20 0)) - значение (10 20 0) - список

“Сливаемые” слоты.

“Сливаемые” слоты отличаются от обычных только методом наследования. В отличие от обычных, “сливаемые” слоты накапливают информацию с различных уровней сети наследования в виде списка. При этом информация с более верхних уровней располагается в этом списке раньше, чем информация с нижних. На порядок следования в результирующем списке влияет и заданный порядок указателей наследования. Для объявления слота как “сливаемого” необходимо объявить его в специальном служебном слоте APP, в том же фрейме. Слияние по уровням вверх продолжается до тех пор, пока на одном из уровней этот ключ не попадает в число объявленых сливаемыми - на этом уровне движение вверх по данной ветви наследования прекращается. Проиллюстрируем механизм вычисления “сливаемых” ключей примерами. Пусть дана некоторая система фреймов, фрагмент которой представлен на рис. G, и пользователь хочет прочесть значение слота ‘XXX из фрейма C1, при этом задана следующая последовательность указателей наследования - UP4, UP2, UP6, в этом случае пользователь получит следующее значение (“Из Б1” “Из Д0” “Из Д1” “Из C1”), легко заметить, что соответствующие значения из фреймов Б0 и Г0 не попали в результирующий список из-за того, что во фрейме Б1, соответствующий ключ не был перечислен в слоте APP. Кроме того следует обратить внимание на тот факт, что порядок следования отдельных элементов совпадает с порядком следования соответствующих указателей наследования, а внутри одной ветви наследования сначала следуют значения из самых верхних фреймов, так Д0 предшествует Д1, в рассматриваемом примере. Необходимо также отметить, что отсутствующие указатели наследования не вносят ни какой информации, в нашем примере, отсутствие на уровне C1 указателя UP6 проявляется так, как будто бы этот указатель наследования не задавался вообще при запросе. Теперь представим себе, что во фрейме Б1, слот XXX объявлен как сливаемый и посмотрим как при этом изменится результат - теперь он будет таким: (“Из Б0” “Из Г0” “Из Б1” “Из Д0” “Из Д1” “Из C1”). В обоих случаях мы предполагаем, что выше фреймов Б0, Г0 и Д0 больше ничего нет.

рис. G



“Разворачиваемые” слоты.

Этот тип слотов добавлен в систему с целью повышения ее эффективности. Строго говоря, это искусственное выделение, без которого система могла бы функционировать. Тем не менее использование этого типа слотов позволяет существенно повысить быстродействие системы. “Разворачиваемые” слоты имеют следующий формат:

(<Ключ_0> & (<Ключ_1> <Ключ_2> ...)), где & - признак “разворачиваемого” ключа, <Ключ_0> - имя этого слота, а (<Ключ_1> <Ключ_2> ...) - список дополнительных слотов. Алгоритм обработки подобных слотов, в общих чертах, выглядит следующим образом:

- возвращается запрошенный ключ (без признака &) -(<Ключ_0> & (<Ключ_1> <Ключ_2> ...));
- определяет значение каждого слота из списка дополнительных ключей - <Ключ_1>, <Ключ_2> и т.д.;
- добавляет все ключи из списка дополнительных к списку “необходимых” (служебный слот ‘_#NEED).

Проиллюстрируем это небольшим примером - пусть дан некий фрейм:

```
(
 (A & (B C P L))
 (C . 15)
 (B . 24.52)
 (P # (L B) (* (+ L B) 2)) )
 (L . 186.78)
 ...
)
```

и мы запросили значение слота A, функция чтения слотов (**@Get-Slots**) вернет результат следующего вида : ((_#NEED A B C P L)(A (B C P L))(B . 24.52)(C . 15)(L . 186.78)(P . 422.6)), таким образом запрос на вычисление слота A вызывает и вычисление всех указанных в нем слотов. Этот прием позволяет уменьшить кол-во обращений к функции чтения слотов. Если бы мы захотели бы проделать тоже, но без использования аппарата “разворачиваемых” слотов то нам необходимо было бы проделать буквально следующее:

```
(setq tmp (@Get-Slots ‘(A) FRM ())
      tmp (cdr (assoc ‘A tmp))
        ext (@Get-Slots tmp FRM ()))
```

вместо

```
(setq ext (@Get-Slots ‘(A) FRM ()))
```

Необходимо при этом отметить, что чем большее число “разворачиваемых” слотов Вы запросили, тем существеннее выигрыш (как по размеру кода, так и по времени) по сравнению с вариантом без использования этого типа слотов.

“Вычисляемые” слоты.

Специальный тип слотов, позволяющий не только значительно сократить объем наполняемых баз, но и существенно расширить их функциональность. В настоящее время формат вычисляемого слота имеет следующий вид:

<вычисляемый слот> ::= “**(**“ <имя> “**#**” <аргументы> <выражение> [<опции>] “**)**”
<аргументы> ::= “**(**“ [<формальные аргументы>] [“*/*” <локальные переменные>] ”**)**”
<формальные аргументы> ::= [<контролируемые аргументы>] [“**ANY:**” <некотролируемые аргументы>] [“**NEED:**” <недостающие характеристики>]
| [<контролируемые аргументы>] [“**NEED:**” <недостающие характеристики>] [“**ANY:**” <некотролируемые аргументы>]
<контролируемые аргументы> ::= *аргументы, которые не могут принимать значение nil при вычислении заданного выражения*
<некотролируемые аргументы> ::= *аргументы, которые могут принимать произвольные значения*
<недостающие характеристики> ::= *характеристики, при отсутсвии которых должен сформироваться запрос на их ввод*
<выражение> ::= *вызов LISP-функции, (если необходимо выполнить более одной ф-ии используйте PROGN)*
<опции> ::= “**(**“ [“**(**“ **LD** <список загрузки> {<список загрузки>} ”**)**”] “**)**”
<список загрузки> ::= “**(**“ <имя функции> <файл> [<путь к файлу>] “**)**”
<имя функции> ::= *LISP-имя функции, которую необходимо подгрузить*;
<файл> ::= *имя файла (с раширением, если не .LSP), в котором должна находиться подгружаемая функция;*
<путь к файлу> ::= *строка или переменная с указание полного либо частичного пути к загружаемому файлу, если не задан загрузка осуществляется по стандартным путям.*

Значение вычисляемого слота зависит от результатов контроля формальных аргументов и режима работы. Нагляднее всего представить результаты в виде небольшой таблицы:

Событие	Режим #@CTRL=2	Режим #@CTRL=1
Контролируемый аргумент принял значение NIL. Один или более.	В качестве значения слота возвращается значение список (#BREAK# <отсутствующие арг.>), кроме того к списку результатов добавляется подсписок (#INTERRUPT# <имя слота> #BREAK# <отсутствующие арг.>).	Выражение не вычисляется, а его значение принимается NIL. Если установлена глобальная переменная #GSLOTS_ALERT, то выдается сообщение с указанием отсутствующих аргументов.
Недостающая характеристика приняла значение NIL. (Группа NEED:)	В качестве значения слота возвращается значение список (#REPEAT# <отсутствующие арг.>), кроме того к списку результатов добавляется подсписок (#INTERRUPT# <имя слота> #REPEAT# <отсутствующие арг.>).	Выражение не вычисляется, а его значение принимается NIL. Если установлена глобальная переменная #GSLOTS_ALERT, то выдается сообщение с указанием отсутствующих аргументов.
Неконтролируемый аргумент принял значение NIL. (Группа ANY:)	Значением слота становится значение вычисленного выражения.	Значением слота становится значение вычисленного выражения.

Представим себе некоторую базу в которой описаны различные окружности, причем нам необходима информация о площади каждой из окружностей. Традиционный метод состоит либо в предварительном вычислении каждой площади и внесении этого параметра в базу данных, либо в написании специальной функции и вычислении этого параметра не через запрос к базе, а путем обращения к этой функции. Оба этих метода не лишены очевидных недостатков. Введение механизма “вычисляемых” слотов (совместно с механизмом наследования) позволяет резко упростить решение этой задачи - вместо конкретного значения для каждого экземпляра базы, на верхнем уровне записывается формула вычисления необходимой величины, так для нашего примера на верхнем уровне достаточно добавить такой слот:
(S # (R) (pi R R)),* где R - имя слота, описывающего радиус окружности,
- признак вычисляемого выражения, *(R)* - список необходимых для вычисления ключей, а *(* pi R R)* - выражение, определяющее значение слота S через указанные параметры (в нашем примере - хорошо знакомое всем со школы πR^2). Поскольку перед R в списке аргументов нет никаких ключевых слов, то перед вычислением выражения будет осуществляться проверка наличия у данного ключа *какого-либо* значения. Контроль типа и значения в настоящее время не осуществляется - а проверяется только факт присутствия этого аргумента. Таким образом, вычисляемое выражение, добавленное во фрейм верхнего уровня, на который ссылаются все элементы базы, позволяет обойтись без указания конкретного значения для каждого элемента. Естественно, это требует также некоторых накладных расходов, таких как введение дополнительного фрейма верхнего уровня (если его еще не было), а также необходимо в каждом элементе базы поставить указатель наследования на этот фрейм. В простейшем случае, как в примере с окружностями, подобная модификация может даже привести к увеличению необходимой памяти, т.к. указатель, обычно, требует больше места чем обычный слот. Однако преимущества даже в этом случае с лихвой окупают некоторое усложнение системы и увеличение объема требуемой памяти. Среди основных достоинств такого подхода необходимо, на наш взгляд, особо остановиться на двух - резкое сокращение объема возможных ошибок при заполнении баз, поскольку верная формула с верными параметрами дает гарантировано верный результат, и более простая модификация, поскольку вместо изменения двух слотов S и R достаточно изменить только один R, а S будет изменяться автоматически. Для реальных баз, у которых, как правило, уже существует верхний уровень и указатели на него уже описаны, к перечисленным достоинствам добавляется еще и существенная экономия памяти, причем она становится тем существеннее, чем большее кол-во “вычисляемых” слотов используется. Кроме явно описанных переменных-слотов выражение может использовать служебных переменную с именем _#FRM - текущий фрейм, для которого запущена функция чтения слотов. Для использования этой переменной внутри вычисляемого выражения нет необходимости ничего предпринимать - эта переменная всегда доступна при вычислении выражения. Естественно, что во избежании ошибок Вы не должны использовать это имя в списке аргументов вычисляемого выражения. Кроме формальных аргументов выражения Вы, также, можете определить локальные переменные, необходимые Вам в процессе вычисления Вашего выражения. Определение локальных переменных аналогично их описанию в определении функций АвтоЛИСПа - все символы в списке аргументов вычисляемого выражения, стоящие после прямой косой черты / воспринимаются как локальные. Например: (WW # (X / XX) (progn (setq xx (* x x)) (/ (sqrt (+ xx xx)) xx))), после вычисления этого выражения значение XX останентся неизменным. Однако локальные переменные не единственный путь решения проблемы сохранения промежуточных результатов вычислений, рассмотрим такой пример:

```
исходный вариант
(Jk # (b Tp h s)
  (if (and b Tp h s)
    (* 4e-5 (+ (* 2 b Tp Tp Tp)
```

```
            (* (- h Tp Tp) s s s)))))
    (Wk # (b Tp h s)
      (if (and b Tp h s)
        (* 4e-4  (/ (+ (* 2 b Tp Tp Tp)
                    (* (- h Tp Tp) s s s)) Tp))))
```

и несколько модифицированный вариант

```
(2bTp3 # (b Tp) (if (and b Tp) (* 2 b Tp Tp Tp))
(h2Tp # (h Tp) (if (and h Tp) (- h Tp Tp))
(s3 # (s) (if s (* s s s))
(Jk # (2bTp3 h2Tp s3) (if (and 2bTp3 h2Tp s3) (* 4e-5 (+ 2bTp3 (* h2Tp s3)))))
(Wk # (2bTp3 h2Tp Tp s3) (if (and 2bTp3 h2Tp Tp s3)(* 4e-4  (/ (+ 2bTp3 (* h2Tp s3)) Tp))))
```

Модифицированный вариант может показаться несколько усложненным, однако в тех случаях, когда пользователь одновременно запрашивает оба ключа: Jk и Wk объем необходимых вычислений меньше, чем в исходном варианте. Это происходит за счет выделения общих частей обеих выражений в отдельные подвыражения, которые вычисляются только один раз. Выигрыш в производительности тем выше, чем сложнее вычисления и чем в большем кол-ве выражений вы можете использовать это подвыражение. При составлении вычисляемых выражений желательно исходить из тех предположений, что все характеристики будут запрошенны одновременно и выделение всех общих подвыражений может внести существенное ускорение вычислений.

При составлении выражения необходимо помнить, что месторасположения вычисляемого слота, обычно не совпадает с местом его запуска. Необходимо достаточно четко себе представлять откуда (с какого уровня) будет запускаться то или иное выражение и какие слоты при этом для него потенциально доступны. Необходимо помнить, что вся информация расположенная ниже уровня запуска является недоступной, в то время как информация с верхних уровней может быть доступна при помощи механизмов наследования. Правильное задание указателей наследования может иметь для вычисляемых слотов очень важное значение, поскольку если выражение использует информацию более высоких уровней, то незаданный указатель наследования может привести к тому, что этот параметр не будет найден и выражение не будет правильно вычислено. Особо внимательно необходимо следить за указателями при множественном наследовании. Необходимо также помнить, что помимо описания указателей наследования внутри фреймов, необходимо указать функции чтения слотов с какими именно указателями наследования ее необходимо вызвать. К сожалению, в настоящее время ответственность за согласованность всех этих параметров ложится на разработчика. Так же разработчик, в настоящее время вынужден следить за отсутствием рекурсивных вызовов (как прямых, так и косвенных) в вычисляемых выражениях. Однако, несмотря на все негативные моменты нынешней реализации, “вычисляемые” выражения существенно расширяют возможности системы.

“Демоны”.

“Демоны” являются, по существу, особой разновидностью вычисляемых слотов, несмотря на ряд отличий, многие особенности у них общие. “Демоны” “следят” за изменениями вносимыми в состав фреймов с помощью функции посылки сообщений (**@SEND**). Каждый “демон” следит за изменением состояния внутри определенного набора слотов, от которых зависит его значение. Один и тот же слот может контролироваться различными “демонами”, в этом случае его изменение приводит к запуску всех этих “демонов”. Результатом работы “демона” обычно служит некоторый побочный эффект: генерация изображения, вывод какой-либо информации, модификация какого-либо фрейма, включая обрабатываемый. Благодаря модификации фреймов путем посылки сообщений “демон” может активизировать запуск других “демонов”. Таким образом появляется возможность разбить задачу на отдельные взаимодействующие подзадачи. Такой метод объединения подзадач гораздо эффективнее и проще стандартного, так как отдельные подзадачи лучше изолированы друг от друга. “Демоны” наиболее мощный механизм в системе. Формат “демонов” весьма напоминает формат “вычисляемых” выражений и выглядит следующим образом:

<демон> ::= “**(**“ <имя> “**@**” <аргументы> <выражение> [<опции>] “**)**”
<аргументы> ::= “**(**“ [<формальные аргументы>] [“**/**” <локальные переменные>] ”**)**”
<формальные аргументы> ::= [<контролируемые аргументы>] [“**ANY:**” <некотролируемые аргументы>] [“**NEED:**” <недостающие характеристики>]
| [<контролируемые аргументы>] [“**NEED:**” <недостающие характеристики>] [“**ANY:**” <некотролируемые аргументы>]
<контролируемые аргументы> ::= *аргументы, которые не могут принимать значение nil при вычислении заданного выражения*
<некотролируемые аргументы> ::= *аргументы, которые могут принимать произвольные значения*
<недостающие характеристики> ::= *характеристики, при отсутсвии которых должен сформироваться запрос на их ввод*
<выражение> ::= *вызов LISP-функции, (если необходимо выполнить более одной ф-ии используйте PROGN)*
<опции> ::= “**(**“ [“**(**“ “**LD**” <список загрузки> {<список загрузки>} ”**)**”] “**)**”
<список загрузки> ::= “**(**“ <имя функции> <файл> [<путь к файлу>] “**)**”

<имя функции> ::= *LISP-имя функции, которую необходимо подгрузить;*
<файл> ::= *имя файла (с раширением, если не .LSP), в котором должна находиться подгружаемая функция;*
<путь к файлу> ::= *строка или переменная с указание полного либо частичного пути к загружаемому файлу, если не задан загрузка осуществляется по стандартным путям.*

Реакции на особые ситуации полностью аналогичны вычисляемым ключам.

Базовые функции.

В системе можно выделить ряд базовых функций, обеспечивающих функционирование всей системы в целом. Эти функции можно условно разбить на несколько разделов, описание которых приведено ниже.

Функции работы со списками.

К функциям ядра необходимо отнести все функции, помогающие реализовать работу с объектами, представленными ЛИСП-списками.

(#Fml <*что нужно*> <*семейство*> <*опции*>)

Файл: *main/fun/mmain.bi4*

Данная функция служит для обработки "семейств" входящих в состав сложных фреймов. Функция позволяет получить либо конкретный фрейм семейства, либо все семейство, либо его произвольную часть. Параметр <*что нужно*> указывает что делать с семейством, существует три варианта значений этого параметра:

Запрос	Результат
ALL	возвращает список всех элементов семейства, предварительно добавив к каждому служебный (0-ой элемент), подробнее см. структуру слота 'FML.
CUT	аналогично 'ALL, но только элементы удовлетворяющие условию заданному слотом 'CUT, либо nil - если таких элементов нет. Если слотом 'CUT не определен, то возвращается все элементы, как и при 'ALL.
(NM . <XXXX>)	возвращается элемент с заданным именем, если такой есть и nil - в противном случае.

Параметр <*семейство*> задает собственно обрабатываемое семейство, так как если оно было бы получено следующим путем *(cdr (assoc 'FML <фрейм>))*. Параметр <*опции*> задает некоторые параметры обработки семейства. Этот параметр имеет формат ассоциативного списка. В настоящее время функция понимает и обрабатывает следующие опции (ключи):

Опция	Действие
FIND	T - вычислять слоты 'FIND если они встретились, nil - игнорировать
FR	T - обрабатывать слоты 'FR если они встретились, nil - игнорировать
SORT	T - сортировать семейство если есть слот 'SORT, nil - игнорировать

ПРИМЕЧАНИЕ. Опции 'FR и 'FIND не работают при запросах 'ALL и 'CUT. При вычислении выражения FIND необходимо помнить что, данное выражение может использовать переменную с именем ACE_NM_TEST. При запуске данной ф-ии из **@FIND_FR** эта переменная уже установлена (локальная в функции **@FIND_FR**), однако в других случаях необходимо устанавливать ее самостоятельно. Для профилактики возможных осложнений рекомендуется придерживаться следующих правил связанных с этой переменной:

- В вызывающей программе ACE_NM_TEST должна быть описана либо как формальный аргумент, либо как локальная переменная.
- Считается, что ACE_NM_TEST - строка, представляющая имя искомого элемента.
- Последнее присвоение значения переменной ACE_NM_TEST перед вызовом данной функции должно осуществляться с помощью ф-ии **SET**, а не **SETQ**.

(#GetFr <*фрейм*>)

Файл: *main/fun/mmain.bi4*

Вычисляет значение слота 'FR заданного фрейма и возвращает найденный фрейм в качестве результата своей работы. Функция добавляет, в случае необходимости, служебный слот с именем 'EXT_NM в найденный фрейм. Значением этого слота является символ - имя переменной, под которым этот фрейм доступен в Автолиспе, или, иными словами, внешнее имя фрейма. Если фрейм не имеет внешнего имени, то значением этого слота является служебный символ '_#CUR. Для найденного фрейма функция выполняет также обработку системного демона 'BEFORE. Аргумент <*фрейм*> должен быть ассоциативным списком или переменной значением которой является такой список. В настоящее время функция распознает два варианта слота 'FR - прямая ссылка на фрейм и поиск заданного фрейма.

Прямая ссылка может быть использована как при задании слотов наследования, так и при указании фрейма в семействе. Механизм с использованием поиска, в настоящее время, используется только при задании слотов наследования. Для успешной обработки слота ‘FR требуется одновременное задание слота ‘NM, а в случае прямой ссылки, и слота ‘PT. Подробнее ознакомиться с форматом соответствующих слотов можно в описании механизма наследования.

(#GetNeed *<а-список> <ключ выборки>* **)**

Файл: *main/fun/mmain.bi4*

Данная функция позволяет выбрать из заданного ассоциативного списка, только те ключи, которые перечислены в указанном параметре. ***<а-список>*** - обычный ассоциативный список; *<ключ выборки>* - имя ключа одного из элементов ассоциативного списка, заданного первым параметром, значением которого является список имен ключей, которые необходимо отобрать.

Результатом работы этой функции является подсписок исходного списка в который включены только те элементы, имена которых перечислены в списке, заданном вторым аргументом функции.

Функция была написана специально для работы в паре с функцией **@Get-Slots**. Если Вы используете функцию **@Get-Slots**, чтобы выполнить какие-либо автоматические действия, то “лишние” (не запрошенные Вами) слоты, не должны оказать никакого влияния на работу программы. Несколько иначе обстоит дело в том случае, когда Вы намереваетесь использовать результат вызова ф-ии **@Get-Slots** для организации взаимодействия с пользователем (например - вывод справки). В этом случае становится трудно найти необходимую информацию среди “мусора”. Чтобы отфильтровать весь мусор Вы можете воспользоваться данной функцией.

Примеры:

Пусть

```
(setq a0 ‘((NM . “A0”)(S # (L B)(* L B))(V # (H S)(* V H))(SV @ (S V))))
(setq a1 ‘((NM . “A1”)(B . 15)(Color . “Красный”)(H . 40)(L . 5)(UP (Nm . “A0”)(Fr . A0))))
(setq b1 (@Get-Slots ‘(Color Sh) A1 ()))
```

тогда

```
(#GetNeed b1 ‘_#NEED) -> ((Color . “Красный”)(SV S V) (S . 75) (V . 300))
(#GetNeed b1 ‘SV)-> ((S . 75) (V . 300))
```

В первом примере на продемонстрирован способ получения списка только запрошенных слотов путем использования в качестве фильтра специализированного служебного слота, а во втором - использование произвольного слота в качестве фильтра.

(#ListAP *<элемент> <сообщение>* **)**

Файл: *main/fun/mmain.bi4*

Данная функция проверяет (так же как и ф-ия **@aList?**) является ли первый аргумент допустимым ассоциативным списком? Если нет то выводится указанное сообщение. Результатом функции является T - если, тестируемый аргумент является ассоциативным списком, и nil - во всех остальных случаях.

Примечание. Сообщение выводится с помощью ЛИСП-функции **PRINC** без всякой обработки (как есть).

Примеры:

```
(#ListAP 1 “Не а-список”) -> nil и печатает: Не а-список
(#ListAP 3.14 “Не а-список”) -> nil и печатает: Не а-список
(#ListAP “String” “Не а-список”) -> nil и печатает: Не а-список
(#ListAP ‘(1 2 3 4) “Не а-список”) -> nil и печатает: Не а-список
(#ListAP ‘((1 2) 3 4) “Не а-список”) -> nil и печатает: Не а-список
(#ListAP ‘((1 2)(3 4)) “Не а-список”) -> T
(#ListAP ‘((1 2) nil (3 4)) “Не а-список”) -> nil и печатает: Не а-список
```

(#Slt+ *<список> <новый элемент>* **)**

Файл: *main/fun/mmain.bi4*

Добавляет в заданный список новый элемент. Новый элемент помещается сразу же после первого элемента исходного списка. Эту функцию удобно использовать для добавления новых слотов во фрейм, так как она пишет новые слоты в начало (переопределяя тем самым старые значения) и сохраняет при этом неизменным первый элемент (имя фрейма).

Примеры:

Пусть

```
(setq a1 ‘(a b c d))
(setq a2 ‘((nm . “abcd)(h . 555)(b . 12))
```

тогда

(#Slt+ a1 555) -> (a 555 b c d)
(#Slt+ a2 '(b . 16)) ->'((nm . "abcd)(h . 555)(b . 16))
(#Slt+ 51 555) -> Ошибка, 51 - не список.

ПРИМЕЧАНИЕ. Функция не контролирует тип своих аргументов!

(#Up_Lmt *<фрейм>*)

Файл: *main/fun/@_ace.bi4*

Функция возвращает список задействованных путей наследования определенных в данном фрейме. Если функция находит в данном фрейме (без учета наследования) слот с именем 'order_up, то значение этого слота используется в качестве списка указателей наследования, в противном случае список берется в ключе ACE_UP (с учетом наследования). При наличии только основного указателя наследования UP функция возвращает nil.

Примеры:

Пусть
(setq a1 '((nm . "kolona_1")(UP ...)(UP4 ...)(UP2 ...) ...)))
(setq a2 '((nm . "kolona_2")(UP ...) ...)))
(setq a3 '((nm . "kolona_1")(UP ...)(UP4 ...)(UP2 ...) (order_up up4 up up2)...)))
и в ACE_ROOT определен слот (ACE_UP UP UP2 UP4 UP6 ...)

тогда
(#Up_Lmt a1) -> (UP UP2 UP4)
(#Up_Lmt a2) -> nil
(#Up_Lmt a1) -> (UP4 UP UP2)

ПРИМЕЧАНИЕ. Обратите внимание на порядок следования указателей - он совпадает с порядком внутри слота order_up или слота ACE_UP.

(@&aLst *<а-список>* *<изменения>*)

Файл: *main/fun/mmain.bi4*

Объединяет два ассоциативных списка. Элемент *<а-список>* содержит исходный ассоциативный список, а *<изменения>* - ассоциативный список вносимых изменений. Элементы второго аргумента перекрывают элементы первого с одинаковыми ключами. Первый элемент исходного а-списка (или, при его замещении, элемент с таким-же ключом) ВСЕГДА сохраняет свою позицию. Если какой-либо из аргументов не задан (имеет значение nil) в качестве значения функции принимается другой, в случае если оба аргумента nil, то результат функции также nil. Функция может нарушать порядок следования элементов внутри ассоциативного списка.

В Н И М А Н И Е !!!

Функция не проверяет являются ли ее аргументы ассоциативными списками. Задание в качестве одного из параметров не ассоциативного списка может привести к ошибке. Рекомендуется выполнять проверку передаваемых аргументов функцией **@aList?** или **#ListAP**.

Примеры:

(@&aLst '((1 . 10)(2 . 20)(3 . 30)) '((3 . 33)(4 . 44))) -> ((1 . 10)(4 . 44)(3 . 33)(2 . 20))
(@&aLst '((1 . 10)(2 . 20)(3 . 30)) '((3 . 33)(1 . 11))) -> ((1 . 11)(3 . 33)(2 . 20))
(@&aLst '((1 . 10)(2 . 20)(3 . 30)) nil) -> ((1 . 10)(2 . 20)(3 . 30))
(@&aLst nil '((3 . 33)(1 . 11))) -> '((3 . 33)(1 . 11))
(@&aLst nil nil) -> nil
(@&aLst '((1 . 10)(2 . 20)(3 . 30)) '(3 . 33)) -> Ошибка, (3 . 33) - не ассоциативный список
(@&aLst '(3 . 30) '((1 . 11)(2 . 22)(3 . 33))) -> Ошибка, (3 . 30) - не ассоциативный список

(@_Ace *<фрейм>* *<управление>*)

Файл: *main/fun/@_ace.bi4*

Функция монитора ACE-технологии обеспечивает интерфейс пользователя с объектом-фреймом.

<фрейм> - возможные значения:

Значение	Описание
'ent	монитор организует запрос на указание примитива и, если указанный примитив - ACE-объект, он преобразуется в фрейм
<имя>	имя примитива, то же, что и 'ent но без запроса
nil	только для рабочего режима. Если в фрейме ACE_SET установлен текущий фрейм, он используется, иначе принимается корневой фрейм ACE_ROOT

A-список	используется в качестве обрабатываемого фрейма
символ	если значение A-список

<управление> - ассоциативный список управления монитором в котором могут быть следующие ключи:

- ограничение наследований (up up2 up4 ..)
- режим справки для указанного фрейма (#select# . 1)
- режим выбора в области, ограниченной выражением (#select# # (список ключей) (выражение))
- текстовый запрос при наличии ключа '#select# - (z_txt (txt ...) (lng ...) ...)

(@aList? *<элемент>*)

Файл: *main/fun/mmain.bi4*

Проверяет является ли аргумент допустимым ассоциативным списком? Если *<элемент>* является ассоциативным списком, то функция возвращает значение T, и nil во всех других случаях.

Примеры:

```
(@aList? nil) -> nil
(@aList? 1) -> nil
(@aList? 3.14) -> nil
(@aList? “String”) -> nil
(@aList? ‘(1 2 3 4)) -> nil
(@aList? ‘((1 2) 3 4)) -> nil
(@aList? ‘((1 2)(3 4))) -> T
(@aList? ‘((1 2) nil (3 4))) -> nil
```

(@Find_Fr *<запрос>* *<управление>*)

Файл: *main/fun/@find_fr.bi4*

Данная функция выполняет поиск фрейма по заданным критериям. В настоящее время реализованы различные виды запросов - от простейшего запроса по внутреннему имени фрейма, до сложных запросов на соответствие заданному критерию. В общем виде структуру любого запроса можно представить в виде следующего списка:

([<критерий поиска>] <имя> <имя среды>), где

<критерий поиска> может представлять собой произвольный безымянный фрейм с обязательным наличием в нем вычисляемого слота ‘FIND, остальное содержимое фрейма совершенно произвольно;

<имя> - это указание полного (nm . “dwtb_16b1”) или группового (nm . “dwtb”) имени искомого объекта;

<имя среды> - указание имени среды (nm . “*SRTM*”), областью которой ограничен данный запрос.

Аргумент <управление> является ассоциативным списком и может содержать следующие ключи:

Ключ	Действие
FIND	имя ключа теста на сложный запрос по умолчанию 'FIND
FIND_N	сколько отбирать при сложном запросе отрицательное значение - искать с конца, при nil,1,-1 возвращается фрейм, в остальных случаях - список фреймов.
FR	t - разворачивать найденный фрейм, nil - вернуть как есть.
GET	t - при неудачном поиске по имени выбор из монитора
HLP	t - дополнительно дать справку на экран
UP	список указателей наследования, используемых при вычислении критерия поиска, если он не задан то используются все имеющиеся в данном фрейме указатели.

Примеры:

поиск по имени

```
(@find_fr '((nm . "dwtb_16b1") (nm . "*SRTM*")) nil)
```

то же с включением справки

```
(@find_fr '((nm . "dwtb_16b1") (nm . "*SRTM*")) '((hlp . t))))
```

поиск по сложному запросу с возвратом развернутого фрейма

```
(@find_fr
 '(((find # (Wx) (> Wx 130))) ;< элемент запроса A-список
   (nm . "krug")
   (nm . "*SRTM*")
  )
 '((hlp . t)))
)
```

то же, но сложный запрос с критерием вне области поиска

```
(@find_fr
```

```
'(((find # (sn1) (and sn1 (< sn1 1000)))
   (up2 (nm . "RS00") (fr fr_0001 "LLLLL")))
  (nm . "krug")
  (nm . "*SRTM*")
 )
'((up up up2)) )
```

Данный пример характерен тем, что в качестве критерия поиска используется некая величина (напряжение sn1), определяемая внутри фрейма *fr_0001* с использованием ключей базы поиска. При этом сам фрейм *fr_0001*

```
(setq fr_0001 '((nm . "RS00")
                (n . 120)
                (Mx . 120.9)
                (sn1 # (n Mx A Wx) (+ (/ n a) (/ Mx Wx)))
                 ...
               ))
```

не является областью базы поиска и весь эффект достигается всего лишь присутствием в запросе слота наследования 'UP2. Если бы в выражении участвовали сотни ключей из базы и фрейма, наследуемого по 'UP2 сама структура запроса не потребовала бы ни каких изменения.

(@Get-Slots <*список слотов*> <*фрейм*> <*опции*>)

Файл: *main/fun/mmain.bi4*

Функция **@Get-Slots** предназначена для чтения указанного списка слотов из заданного фрейма. Функция реализует описанные механизмы наследования, а также обработку различных типов слотов (вычисляемые, разворачиваемые, сливаемые). <*Список слотов*> представляет собой "линейный" (одноуровневый) список имен слотов, значения которых необходимо прочесть, этот аргумент не может быть атомом; <*фрейм*> - представляет собой ассоциативный список с которого начинается поиск указанных слотов, должен быть ассоциативный либо пустой список или его имя; <*опции*> - ассоциативный список управляющих параметров, в настоящее время обрабатываются следующие опции:

Ключ	Формат	Описание
UP	список	Список актуальных указателей наследования в порядке их приоритетов. Если ничего не задано, подразумевается использование указателя наследования по умолчания - UP. Например: (UP UP8 UP2 UP4 UP UP7), обратите внимание что первый символ UP в списке является просто ключем группы опций, а второй указателем наследования!
#@CTRL	0, 1 или 2.	Режим вычисления слотов-выражений и демонов: 0 - ни выражения, ни демоны не вычислять; 1 - выражения вычислять, демоны - нет; 2 - выражения и демоны вычислять. По умолчанию действует режим 1.

Необходимо обратить внимание на тот факт, что функция использует указатели наследования в том порядке в котором они заданы, что в некоторых случаях может иметь существенное значение. В настоящее время существует специальный указатель наследования, используемый функцией **@Get-Slots** в любом случае, даже если он не задан явно - в этом случае он добавляется в конец списка явно заданных указателей. Этот специальный указатель имеет имя 'UP. Даже если при вызове функции список опции был пуст, либо в нем не было списка указателей наследования, то функция все равно будет наследовать информацию по специальному указателю 'UP.

Результатом работы функции является ассоциативный список всех запрошенных слотов, а также всех слотов, значения которых понадобились для вычисления запрошенных. Порядок слотов в результирующем списке, на сегодняшний день, не определен. Кроме того в результирующий список записывается специальный слот '_#NEED в котором перечислены имена всех слотов, которые были запрошены, а также имена слотов которые были указаны в качестве параметров "разворачиваемых" слотов. Вы можете воспользоваться слотом _#NEED для отсева ненужных слотов (в частности - при выводе на экран, при записи в файл и т.п.).

Примеры:

Пусть

```
(setq a0 '((NM . "A0")(S # (L B)(* L B))(V # (H S)(* V H))(SV @ (S V))))
(setq a1 '((NM . "A1")(B . 15)(Color . "Красный")(H . 40)(L . 5)(UP (Nm . "A0")(Fr . A0))))
```

тогда

(@Get-Slots '(Color Sh) A1 nil) вернет следующий список:
((B . 15)(H . 40)(L . 5)(SV S V)(S . 75)(V . 300)(Color . "Красный")(_#NEED Color SV S V))

ЗАМЕЧАНИЯ. Как легко заметить, в результате работы функции образуется ассоциативный список, причем ни для какого ключа нельзя установить ни его тип (простой, вычисляемый или разворачиваемый, наследуемый или сливаемый, ни его реальное местонахождение (в каком из узлов иерархического дерева определен тот или иной ключ). Все слоты результирующего списка имеют абсолютно одинаковый формат - *(<ключ> . <значение>).*

(@Only1st *<а-список> <удаления>*)

Файл: *main/fun/mmain.bi4*

Удаляет из ассоциативного списка *<а-список>* все вхождения элементов (кроме первых) с одинаковыми ключами. Если второй список не пуст, то из списка удаляются ВСЕ вхождения элементов с указанными в этом списке ключами. Аргумент <удаления> должен быть либо nil, либо простым линейным списком.

В Н И М А Н И Е !!!

Функция не контролирует тип своих аргументов. Рекомендуется выполнять соответствующие проверки.

Примеры:

Пусть
(setq a '((1 . 10)(2 . 20)(1 . 11)(2 . 22)(3 . 30)(5 . 55)))

тогда
(@Only1st a nil) -> ((1 . 10)(2 . 20)(3 . 30)(5 . 55))
(@Only1st a '(2 3)) -> ((1 . 10)(5 . 55))
(@Only1st nil '(2 3)) -> nil
(@Only1st '(1 2 3 4 5) nil) -> Ошибка, (1 2 3 4 5) - не ассоциативный список
(@Only1st a 2) -> Ошибка, 2 - не список

(@raLst *<а-список> <удаления>*)

Файл: *main/fun/mmain.bi4*

Удаляет из аргумента *<а-список>* все элементы ключи которых перечислены в аргументе *<удаления>*. Аргумент *<а-список>* должен быть ассоциативным списком, а аргумент *<удаления>* - простой (линейный) список ключей. Если в качестве второго аргумента задан пустой список, то результатом будет исходный ассоциативный список без каких-либо изменений.

В Н И М А Н И Е !!!

Функция не контролирует тип своих аргументов. Рекомендуется выполнять соответствующие проверки.

Примеры:

(@raLst '((1 . 10)(2 . 20)(3 . 30)) nil) -> ((1 . 10)(2 . 20)(3 . 30))
(@raLst '((1 . 10)(2 . 20)(3 . 30)) '(2 4)) -> ((1 . 10)(3 . 30))
(@raLst '((1 . 10)(2 . 20)(3 . 30)) 2)) -> Ошибка, 2 - не список

(@Send *<список ключей> <фрейм> <управление>*)

Файл: *main/fun/@send.bi4*

Данная функция обеспечивает запись списка слотов во фрейм с обработкой запуска актуальных демонов.

Функция возвращает одно из трех значений :

Значение	Описание
'#ok#	<список ключей> записан, актуальные демоны отработали корректно
'#break#	один или более из актуальных демонов вернул значение '#break#, в связи с чем <список ключей> не записан, состояния фреймов и AUTOCADa возвращены в точку запуска **@Send**
#repeat#	один или более из актуальных демонов вернул значение '#repeat#, в связи с чем <список ключей> не записан, состояния фреймов и AUTOCADa возвращены в точку запуска **@Send**
nil	<список ключей> записан, актуальных демонов не найдены

<список ключей> - ассоциативный список для записи во фрейм, например - '((a . v1)(b . v2) (c # (f b) (* f b)));

<фрейм> - фрейм - куда записывать ключи, ассоциативный список, либо его символ-внешнее имя, например 'ACE_ENV;

<управление> - аргумент является ассоциативным списком и может содержать следующие слоты : (up up2 ...) - пути наследования; (ev_key . t) - записываемые ключи поместить во фрейм в вычисленном виде.

Функция среди демонов данного фрейма (ключ 'KEY_DEMON) отбирает актуальные, т. е. содержащие в своем списке аргументов хотя бы один ключ из <списка ключей>. Создается служебный фрейм, куда входят актуальные демоны, преобразованные к форме "обычных вычисляемых ключей и искусственный слот наследования на <фрейм>, после чего актуальные демоны вычисляются функцией **@Get-Slots**. Пути наследования, при этом определяются следующим образом:

- если в списке управления есть слот 'UP - он используется
- иначе автоматически определяются все имеющиеся в фрейме пути (из перечисленных в ACE_ROOT-е)

Если хотя бы один из демонов при вычислении возвращает '#break# дальнейшее действие функции **@Get-Slots** прерывается, исходное состояние фреймов и AUTOCADa восстанавливается и функция **@Send** возвращает '#break#

Иначе, если хотя бы один из демонов при вычислении возвращает '#repeat#, исходное состояние фреймов и AUTOCADa восстанавливается, и повторяются все действия функции.

Структура демона и его служебного списка описана в разделе "Демоны".

(Get-Val <*ключ*> <*a-список*>)

Файл: *main/fun/mmain.bi4*

Функция возвращает НЕ ВЫЧИСЛЕННОЕ (как есть) значение указанного ключа в заданном ассоциативном списке.

Примеры:

Пусть
```
(setq a '((1 . 10)(2 20)(3 30 31 32)(4 . "one string")(5 # (B H)(* B H)(6))
```

тогда
```
(Get-Val 1 a) -> 10
(Get-Val 2 a) -> (20)
(Get-Val 3 a) -> (30 31 32)
(Get-Val 4 a) -> "one string"
(Get-Val 5 a) -> (# (B H)(* B H))
(Get-Val 6 a) -> nil
(Get-Val 7 a) -> nil
```

ПРИМЕЧАНИЕ. Обратите внимание на тот факт, что с помощью данной функции нельзя отличить отсутствует ли заданный ключ вообще в списке, либо он есть но его значение nil (см. примеры с ключами 6 и 7). Результат работы этой функции полностью идентичен выражению *(cdr (assoc <ключ> <a-список>))*.

Функции работы с графикой.

Общие положения.

В этом разделе описываются функции переноса фрейма в *DWG*-файл и создания изображений ACE-объектов. В Автокаде ACE-объект - это блок с уникальным именем и с расширенными данными. ACE-объекты могут быть простые и сложные. Простые ACE-объекты - это объекты, в состав которых не входят другие ACE-объекты (например, труба, заклепка и т.п.). Сложные ACE-объекты - это объекты, в состав которых входят другие ACE-объекты.

В общем случае ACE-объекты могут иметь как трехмерные изображения, так и плоские (виды, разрезы). Здесь будем рассматривать описание ACE-объектов в прямоугольной декартовой системе координат. Система называется прямоугольной декартовой, если оси координат взаимно перпендикулярны, а единичные отрезки на осях равны. Система видов и разрезов ортогональная. Направления видов совпадают с осями системы координат. Плоскости разрезов перпендикулярны направлениям видов.

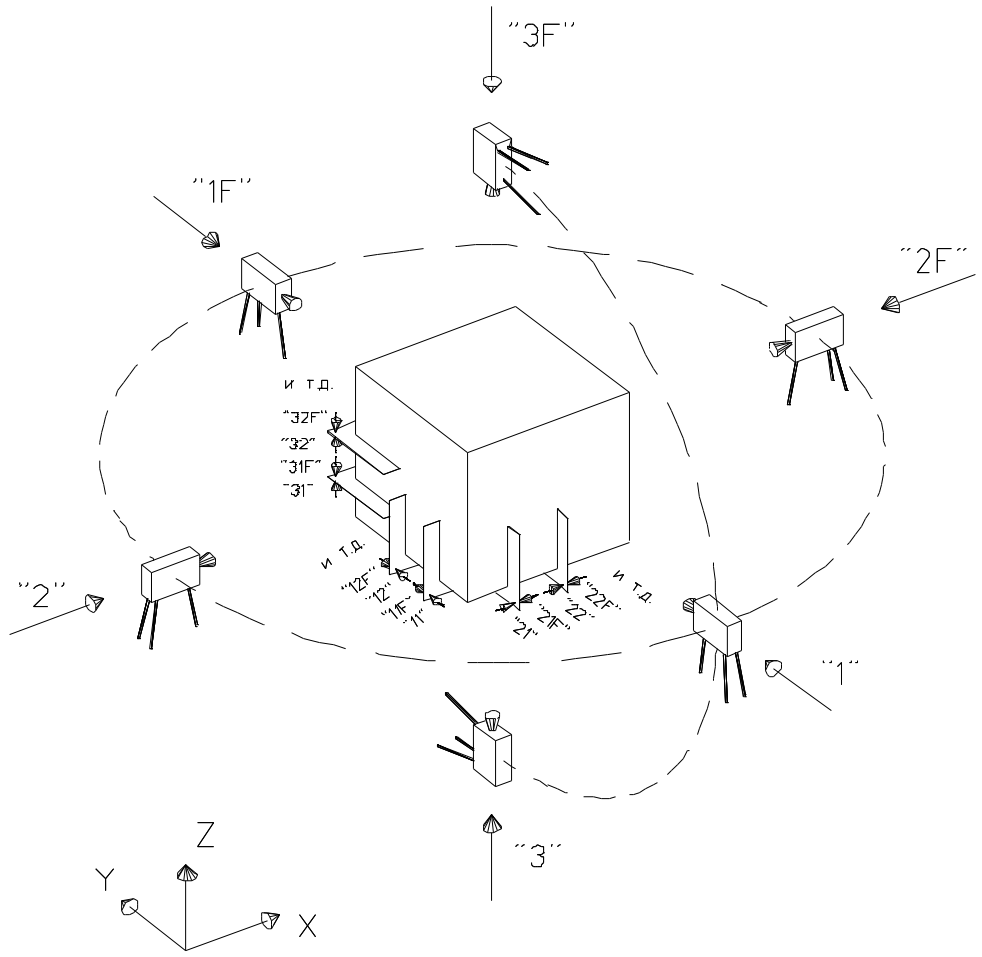


рис. H

Предусмотрено 6 видов: три прямых - “1”, “2” и “3” (см. рис. H) и три обратных - “1F”, “2F” и “3F”.

Имена разрезов определяются последовательно передвигаясь в направлении прямых видов “1”, “2” и “3”.

В ACE-среде предусматривается возможность отрисовывать объекты с различной степенью детализации. Всего может быть пять степеней детализации:

Степень детализации	Краткое описание
“1”	Наиболее полное, без упрощений изображение ACE-объекта.
“2”	Допускаются некоторые упрощения в изображении ACE-объекта.
“3”	Упрощенное изображение ACE-объекта.
“4”	Изображение ACE-объекта в виде объемлющего тела. Отрисовывается параллелепипед или цилиндр для трехмерных изображений и прямоугольник или круг для плоских изображений.
“5”	Отрисовывается условное изображение ACE-объекта (схема).
“6”	Отрисовывается условное изображение ACE-объекта (точка, в том числе - невидимая).

Следует заметить, что в общем случае изображения различных степеней детализации могут совпадать.

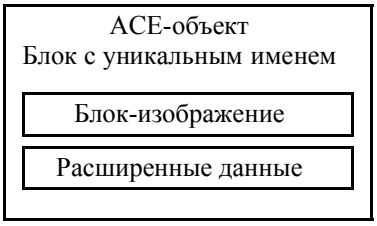
Это были общие замечания не требующие особых знаний. Для понимания информации, которая будет изложена в следующих разделах, следует разобраться с такими понятиями, как:

- Демоны (функция **@Send**).
- Вычисляемые выражения.
- Наследование информации (функция **@Get-Slots**).

Функция переноса фрейма в DWG-файл.

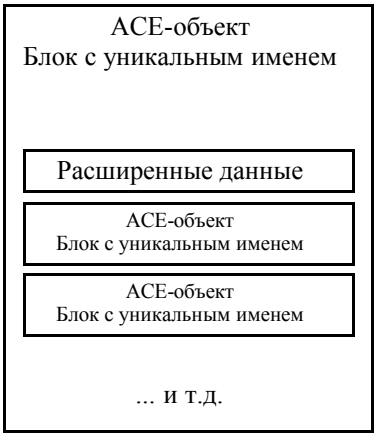
Это универсальная функция. Предназначена для создания любых ACE-объектов. Имя функции **@ACE_IMG** (файл *ace_img.lsp*). В Автокаде ACE-объект - это блок с уникальным именем и с расширенными данными. В зависимости от ситуации возможны три варианта структуры ACE-объектов:

Первый вариант



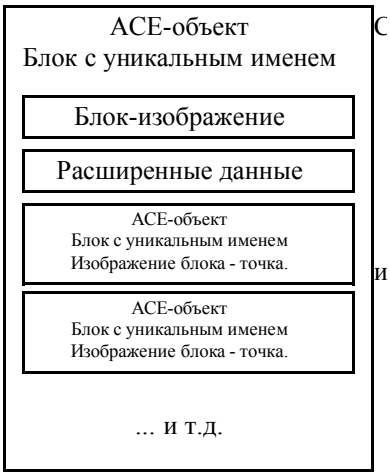
Простые ACE-объекты. В Автокаде это блок с расширенными данными в состав которого входит блок-изображение.

Второй вариант



Сложные ACE-объекты. ACE-объект описан в виде фрейма, содержащего семейство. В Автокаде это блок с расширенными данными в состав которого могут входить как простые, так и сложные ACE-объекты.

Третий вариант



Сложные ACE-объекты. ACE-объект описан в виде фрейма, содержащего семейство. В Автокаде это блок с расширенными данными. Изображение объекта формируется как один блок-изображение. Все входящие в состав ACE-объекты отрисовываются как точки (точки могут быть невидимыми). Такие ACE-объекты будут формироваться, если в базе будет установлен ключ

(flag_img_obj . t)

и функция создания изображения будет обрабатывать такую ситуацию, т.е. создавать изображение всего сложного ACE-объекта.

При создании сложных ACE-объектов функция **@ACE_IMG** вызывается рекурсивно. Функция выполняет масштабирование и окончательную установку ACE-объекта. Выполняется также запись расширенных данных по ключу XD из текущего фрейма с наследованием. Ниже приводится фрагмент фрейма с описанием ключа XD:

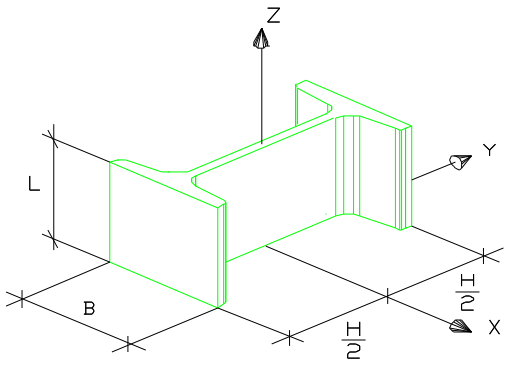
```
...
(XD # (nm l app)
      (XData
        (list
          (list 'up4 (cons 'nm nm)
            '(fr find (nm . "*SRTM*"))
          )
          (cons 'l l)
          (cons 'app app)
        )
      )
```

)

...

Подробнее о способах задания положения ACE-объектов. Функции создания изображений отрисовывают ACE-объекты в исходном положении, т.е. так, как они были описаны в базе. Например, двутавры в базах ACE-технологии расположены относительно локальной системы координат так как изображено на рис. I:

рис. I



Окончательную установку ACE-объекта выполняет функция переноса фрейма в *DWG*-файл. В ACE-технологии предусмотрено два режима работы с объектами - ручной и автоматический. При ручном режиме установки ACE-объектов пользователь в диалоге выбирает способ задания положения и указывает соответствующие значения параметров установки. Для автоматической установки ACE-объекта необходимо правильно и полно определить группу INSERT в текущем фрейме с учетом наследования. Поэтому достаточно хорошо разобраться в структуре группы INSERT, чтобы потом и в ручном режиме работы с ACE-объектами не возникало проблем.

Локальная система координат, в которой описываются ACE-объекты, совпадает с текущей ПСК. Все координаты о которых будем говорить ниже задаются именно в текущей ПСК. Это координаты точек в ключах PT1, PT2, PT3, NEW_PT1, NEW_PT2 и NEW_PT3.

Для трехмерных ACE-объектов предусмотрено три способа задания положения в пространстве:

- Точка и три угла поворота. Структура группы INSERT в этом случае:

Здесь ключ FLAG указывает каким способом задается положение объекта. Это строковая константа. Изменять нельзя. Следующий ключ PT1 - координаты базовой точки на ACE-объекте, который расположен в исходном положении. Ключ NEW_PT1 - новые координаты базовой точки. Ключи NEW_AX, NEW_AY и NEW_AZ задают соответственно углы поворота вокруг осей X, Y и Z. При этом считаем, что начальные значения углов равны нулю (ACE-объект в исходном положении). Углы задают чистый поворот ACE-объекта.

- По трем точкам. Точки PT1, PT2 и PT3, а также NEW_PT1, NEW_PT2 и NEW_PT3 не должны лежать на одной прямой. Структура группы INSERT в этом случае:

Здесь ключ FLAG указывает каким способом задается положение объекта. Это строковая константа. Изменять нельзя. Ключи PT1, PT2 и PT3 задают векторы PT1-PT2 и PT1-PT3 связанные с ACE-объектом в исходном положении. Ключи NEW_PT1, NEW_PT2 и NEW_PT3 задают новое положение векторов. ACE-объект устанавливается в новое положение так, чтобы базовая точка PT1 совпала с точкой NEW_PT1, направление вектора PT1-PT2 совпало с направлением вектора NEW_PT1-NEW_PT2, а векторы PT1-PT3 и NEW_PT1-NEW_PT3 лежали в одной плоскости.

- Две точки и угол поворота вокруг оси, проходящей через эти точки. Структура группы INSERT в этом случае:

Здесь ключ FLAG указывает каким способом задается положение объекта. Это строковая константа. Изменять нельзя. Ключи PT1 и PT2 задают положение вектора, связанного с ACE-объектом в исходном положении. Ключи NEW_PT1 и NEW_PT2 задают новое положение вектора. ACE-объект устанавливается в новое положение так, чтобы базовая точка PT1 совпала с точкой NEW_PT1. Вектор PT1-PT2 совмещается с направлением вектора NEW_PT1-NEW_PT2. Угол поворота в ключе NEW_ANG указывает на сколько необходимо повернуть ACE-объект вокруг совмещенных векторов.

Для плоских ACE-объектов предусмотрено два способа задания положения на плоскости:

- Точка и угол поворота вокруг этой точки. Структура группы INSERT в этом случае:

Здесь ключ FLAG указывает каким способом задается положение объекта. Это строковая константа. Изменять нельзя. Ключ PT1 задает положение базовой точки на ACE-объекте. Ключ NEW_PT1 задает новое положение ACE-объекта на плоскости. Ключ NEW_ANG - угол поворота ACE-объекта вокруг базовой точки. При этом считаем, что начальное значение угла равно нулю (ACE-объект в исходном положении).

- По двум точкам. Структура группы INSERT в этом случае:

Здесь ключ FLAG указывает каким способом задается положение объекта. Это строковая константа. Изменять нельзя. Следующие два ключа PT1 и PT2 задают положение вектора, связанного с ACE-объектом в исходном положении. Ключи NEW_PT1 и NEW_PT2 задают новое положение вектора. ACE-объект переносится в новое положение так, чтобы базовая точка PT1 совпала с точкой NEW_PT1, а направление векторов совпадали.

Функция создания изображения объемлющего тела.

Это универсальная функция создания изображений ACE-объектов. Имя функции **@IMG_ALL** (файл *img_all.lsp*). Для трехмерного режима создания ACE-объектов это либо прямоугольный параллелепипед либо цилиндр вдоль одной из осей, вписанный в прямоугольный параллелепипед. Для режима создания плоских изображений ACE-объектов (виды, разрезы) это либо прямоугольник, либо окружность. В случае отсутствия информации для отрисовки объемлющего тела либо если она не полная, отрисовывается точка (примитив Автокада ТОЧКА).

Функция запускается как демон, активизируемый записью во фрейм ключа DRW_ALL (см. файл *ace_root.fr*). Ниже приводится фрагмент из файла *ace_root.fr* :

Для отрисовки объемлющего тела ACE-объекта в текущем фреиме с учетом наследования должна быть слот с ключем ENCL. Слот читается из текущего фрейма функцией **@Get-Slots** и может быть описан как вычисляемый. В общем случае слот ENCL может содержать 7 слотов с ключами: LX-, LX+, LY-, LY+, LZ-, LZ+ и TYP_APR. Первые шесть слотов описывают размеры и положение объемлющего тела в локальной системе координат. Это шесть обязательных слотов для отрисовки прямоугольного паралелепипеда. Если описаны не все шесть первых слотов, функция будет отрисовывать точку. Для управления отрисовкой объемлющего тела в виде цилиндра предназначен слот с ключем TYP_APR. Если он определен как один из трех описанных ниже вариантов, то объемлющее тело ACE-объекта будет отрисовываться в виде цилиндра, вписанного в прямоугольный параллелепипед следующим образом :

(TYP_APR . “CYL_X”) - ось цилиндра вдоль оси X.
(TYP_APR . “CYL_Y”) - ось цилиндра вдоль оси Y.
(TYP_APR . “CYL_Z”) - ось цилиндра вдоль оси Z.

Ниже приводится фрагмент из фрейма для двутавров. Будет отрисовываться прямоугольный параллелепипед.

```
...
(ENCL # (h l) (list (cons 'LX+ (*  0.5 h))
                    (cons 'LX- (* -0.5 h))
                    (cons 'LY+ (*  0.5 h))
                    (cons 'LY- (* -0.5 h))
                    (cons 'LZ+ l)
                    (cons 'LZ- 0.0)
                    '(TYP_APR . “CYL_Z”)
              )
)
...
```

Пример группы ENCL для отрисовки объемлющего тела в виде цилиндра (фрагмент из файла с описанием базы круглой стали):

Разрезы отрисовываются как прямоугольник или круг, т.е. режется объемлющее тело. Однако любой разрез вдоль цилиндра будет отрисован как прямоугольник по размерам, описанным в слоте ENCL.

В любом случае, если функция вызвана, будет создан блок-изображение:

- прямоугольный параллелепипед - выдавленная на определенную высоту полилиния;
- цилиндр - выдавленный на определенную высоту круг;
- точка - может быть либо видимой, либо невидимой.

В ACE-технологии задачу создания изображений ACE-объектов выполняют две функции : функция создания изображения объемлющего тела **@ACE_ALL** и функция создания изображения ACE-объектов (см. ниже). Функция **@ACE_ALL** - универсальная, управление которой осуществляется через слот ENCL. Поэтому следует помнить, что размеры и положение объемлющего тела должны согласовываться с тем, как расположен объект относительно локальной системы координат (см. функцию создания изображения ACE-объектов).

Функция как демон всегда возвращает код возврата '#ok#.

Функция создания изображения ACE-объектов.

Функция индивидуальная. Предназначена для формирования изображения ACE-объекта и является неотъемлимой частью описания конкретной группы объектов.

Требования к функции:

- При определенном режиме вызова функция должна возвращать информацию о том, какие изображения она способна создавать.
- Для отрисовки ACE-объекта функция вызывается как демон. В режиме вызова как демон функция должна возвращать ключи возврата демонов:

'#ok# - отрисовка прошла нормально.

'#break# - отказ.

- В функции должны быть обязательно использованы ключи IMG и DET.

Значения ключа IMG (строковая):

Ключ	Краткое описание
"D"	отрисовывать трехмерное изображение.
"1"	вид на объект вдоль положительного направления оси Y.
"1F"	вид на объект в направлении, противоположном положительному направлению оси Y.
"2"	вид на объект вдоль положительного направления оси X.
"2F"	вид на объект в направлении, противоположном положительному направлению оси X.
"3"	вид на объект вдоль положительного направления оси Z.
"3F"	вид на объект в направлении, противоположном положительному направлению оси Z.
"11"	вид на плоскость первого разреза вдоль положительного направления оси Y.
"11F"	вид на плоскость первого разреза в направлении, противоположном положительному направлению оси Y.
...	и т.д.
"21"	вид на плоскость первого разреза вдоль положительного направления оси X.
"21F"	вид на плоскость первого разреза в направлении, противоположном положительному направлению оси X.
...	и т.д.
"31"	вид на плоскость первого разреза вдоль положительного направления оси Z.
"31F"	вид на плоскость первого разреза в направлении, противоположном положительному направлению оси Z.
...	и т.д.

Не исключается возможность описания видов под произвольным углом зрения и разрезов по произвольной плоскости. Однако эти ситуации должны соответствующим образом обрабатываться функцией создания изображения...

Значения ключа DET (строковая):

Ключ	Описание
"1"	наиболее полное, без упрощений, изображение ACE-объекта.
"2"	допускаются некоторые упрощения в изображении ACE-объекта.
"3"	упрощенное изображение ACE-объекта.
"4"	в этой функции не использовать. При этой степени детализации вызывается функция отрисовки объемлющего тела (см. выше).
"5"	отрисовывается условное изображение ACE-объекта (схема).
"6"	отрисовывается условное изображение ACE-объекта (точка).

- Любое из отрисовываемых изображений ACE-объекта должно отрисовываться с любыми степенями детализации. В общем случае изображения для различных степеней детализации могут совпадать.
- Окончательно функция должна создать блок-изображение в масштабе 1:1 и вставить его в указанную точку. При этом желательно выполнить проверку - если подходящий блок-изображение уже существует, вставить его, в противном случае создать блок-изображение с соответствующим именем и вставить его в указанную точку.

Подробнее разберемся на примере функции создания изображений для двутавров всех типов.

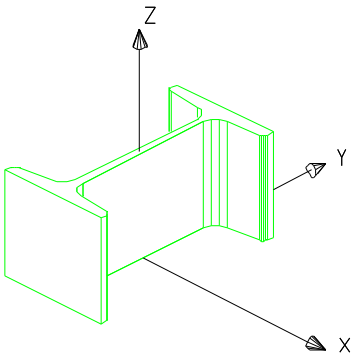


рис. J

Прежде, чем приступать к созданию функции, следует решить, как будет расположен объект относительно локальной системы координат и создать базу объектов. О том, как расположен объект относительно локальной системы координат следует помнить и при описании слота ENCL, управляющего функцией отрисовки объемлющего тела. На рис. J показано как расположены двутавры относительно локальной системы координат.

Фрагмент базы двутавров низкого уровня (файл *dwtr.fr*). Это самый низкий уровень базы. Выделены слоты которые используются функцией создания изображений ACE-объектов.

```
...
 ( (nm   . "DWTR_10")
  (keyf   (nm (txt . "Двутавр 10") (lng Rdwtr  "SRTM")))
  (H  .  100.0)       ; Высота
  (b  .   55.0)       ; Ширина
  (S  .    4.5)       ; Толщина стенки
  (Tp .    7.2)       ; Толщина полки
  (r  .    7.0)       ; Радиус скругления полка-стенка
  (r1 .    2.5)       ; Радиус скругления конца полки
  (a  .   12.0)
  (Jx .  198.0)
  (Sx .   23.0)
  (Jy .   17.9)
 )
 ( (nm   . "DWTR_12")
  (keyf   (nm (txt . "Двутавр 12") (lng Rdwtr  "SRTM")))
  (H    .  120.0)
  (b    .   64.0)
  (S    .    4.8)
  (Tp   .    7.3)
  (r    .    7.5)
  (r1   .    3.0)
  (A    .   14.7)
  (Jx   .  350.0)
  (Sx   .   33.7)
  (Jy   .   27.9)
  (sokr . t)
  (St    C255 C285)
 )
... и т.д.
```

Ниже фрагмент из файла *a_dwt_0.fr* . Вычисляемый слот ENCL содержит информацию о размерах и положении объемлющего тела для двутавров. Здесь нет слота с ключем TYP_APR, следовательно будет отрисовываться объемлющее тело в виде прямоугольного паралелепипеда. Имя функции создания изображений всех типов двутавров **@SRTM_DWT** (файл *srtm_dwt.lsp*). В демоне эта функция вызывается в режиме создания изображений (см. выделенную строку). В вычисляемом слоте LMT_IMG функция **@SRTM_DWT** вызывается в режиме возврата списка имен изображений, которые она может создавать (см. выделенную строку, а также текст функции **@SRTM_DWT**).

```
...
   ; Размеры объемлющего тела.
   (ENCL # (b h l) (list (cons 'LX+ (*  0.5 b))
                        (cons 'LX- (* -0.5 b))
                        (cons 'LY+ (*  0.5 h))
                        (cons 'LY- (* -0.5 h))
                        (cons 'LZ+ l)
                        (cons 'LZ- 0.0)
                  )
   )
   ; Демон отрисовки низкого уровня.
   (drw_dn @
     (drwdn)
     (@srtm_dwt _#FRM)
     (
      (ld (@srtm_dwt "bz0/srtm_dwt" srtm_path)
          (@ace_lcl “fun/ace_lcl” ace_main_pt)
      )
      (typ exit hid)
     )
   )
   ; Список имен изображений, которые может создавать функция создания изображений.
   (lmt_img # ()
     (@srtm_dwt 'lmt)
     ((ld (@srtm_dwt "bz0/srtm_dwt" srtm_path)))
   )
   ...
```

Далее фрагмент из файла *a_srtm_u.fr* . Вычисляемый слот NM_IMG генерации имени блока-изображения для всех ACE-объектов SRTM-среды. Возможно два варианта значений слота :

- *(“Y” <имя блока-изображения>)* - если блок с созданным именем уже создан т.е. есть в таблице блоков;
- *(“N” <имя блока-изображения>)* - блока с созданным именем нет.

```
...
   (nm_img # (ext_nm l det img)
     (progn
      ; Если есть ключ up4, читаю имя из этого ключа. Иначе читаю ключ nm.
      (if (setq _#2 (assoc 'up4 (eval ext_nm)))
       (setq _#4 (cdr (assoc 'nm (cdr _#2))))
       (setq _#4 (cdr (assoc 'nm (eval ext_nm))))
      )
      ; В именах блоков запрещено использовать символы - (минус) и . (точка),
      ; поэтому здесь замещаю в именах эти символы...
      ; Замещаю точку на символ _.
      (setq _#4 (m:rplstr _#4 "." "_"))
      (setq l (rtos# l))
      (setq l (m:rplstr l "." "_"))
      ; Замещаю тире на символ $.
      (setq _#4 (m:rplstr _#4 "-" "$"))
      (setq _#1 (strcat _#4 det img l))
      ; Возвращаем список. Первый элемент списка - строковая либо "Y" такой
      ; блок есть, либо "N" такого блока нет. Второй элемент списка имя
      ; блока-изображения (строка).
      (cond ((tblsearch "block" _#1) (list "Y" _#1))
            (t (list "N" _#1))
      )
     )
     ((ld (m:rplstr “video” gl_path) ; Загрузка функций при необходимости.
          (rtos# “fun/rtos#” ace_main_pt)
     ))
   )
...
```

Распечатка функции создания изображения для двутавров всех типов. Это не окончательный вариант функции. Однако это пример работающей сегодня функции и принципиальных изменений в ней скорее всего не будет.

```
;************************************************************  файл SRTM_DWT.LSP
; ДПИ КМТ Рыбас В.   г.Днепропетровск   (0562) 92-36-47
;*****************************************************************************
;
```

```
; Функция настройки на слой, цвет и тип линии.
(if (not @ace_lcl) (load_1 ace_main_pt "fun/ace_lcl"))

;-------------------------------------------------------------------------------
; Функция отрисовки ДВУТАВРОВ ВСЕХ ТИПОВ.
; Должна формировать все  предусмотренные  изображения.  Отрисовка  выполняется
; в масштабе 1:1. Функция  должна  создать изображение в виде  блока и вставить
; блок в указанную точку PS0. В  этом  случае  возвращать  или  передавать  'вверх'
; ничего не нужно.  Функция переноса фрейма в DWG-файл выполняет масштабирование
; ACE-объекта.
;
; a_list - ассоциативный  список.  Читать  из  списка  функцией  @get-slots для
;          использования возможности наследования информации.
;-------------------------------------------------------------------------------
(defun @srtm_dwt (a_list / b h s tp r1 r uklon det img l lr ls img_set kodw
                           u w x y a c0 c1 c2 c3 c4 c5 c6 t45 ps0
                           tmp tmpf #up nm_img img_h img_s img_l
                 )
 (if (= a_list 'lmt)
  (progn
   ; Если в качестве аргумента передан символ 'lmt, то функция возвращает
   ; список всех типов изображений, которые способна генерировать эта
   ; функция (ключ img). При этом считаем, что все степени детализации
   ; описаны. Приведенный ниже список говорит о том, что эта функция
   ; может создать трехмерное изображение объекта (прокатного двутавра) со
   ; всеми степенями детализации, шесть видов и два разреза также со всеми
   ; степенями детализации.
   '("D" "1" "1F" "2" "2F" "3" "3F" "31" "31F")
  )
  (progn
   ; Читаем информацию функцией @get-slots из a_list. Писать только во фрейм,
   ; имя которого записано в слоте ext_nm (см. ниже tmp и tmpf).
   (setq tmp  (@get-slots '(ext_nm) a_list nil) ; Слот с имнем фрейма.
         tmpf (cdr (assoc 'ext_nm tmp))         ; Символ - имя фрейма.
         #up  (#up_lmt (eval tmpf))  ; Список наследований '((up up4 ...))
   )
   ; Формирую ассоциативный список только с нужными группами.
   (setq lr (@get-slots '(b h s tp r1 r uklon det img l img_set ps0 nm_img)
                        a_list  #up
            )
   )
   ; Присваиваю соответствующим переменным значения...
   (setq ps0    (cdr (assoc 'ps0    lr))
         nm_img (cdr (assoc 'nm_img lr))
   )
   (if (= (car nm_img) "N")
    ; Нужного блока нет. Создаем его...
    (progn
     ; Для удобства создания изображения...
     (setq b       (cdr (assoc 'b       lr))
           h       (cdr (assoc 'h       lr))
           s       (cdr (assoc 's       lr))
           tp      (cdr (assoc 'tp      lr))
           r1      (cdr (assoc 'r1      lr))
           r       (cdr (assoc 'r       lr))
           uklon   (cdr (assoc 'uklon   lr))
           det     (cdr (assoc 'det     lr))
           img     (cdr (assoc 'img     lr))
           l       (cdr (assoc 'l       lr))
           img_set (cdr (assoc 'img_set lr))
           b  (* b 0.5)
           h  (* h 0.5)
           s  (* s 0.5)
           r1 (if r1 r1 (* tp 0.1))
           u  (if uklon (/ uklon 100.0) 0.0) ; Тангенс угла наклона полки.
           ; Информация для настройки на слои, цвета и типы линий...
           img_l (cdr (assoc 'lin_l img_set))
           img_s (cdr (assoc 'lin_s img_set))
           img_h (cdr (assoc 'lin_h img_set))
     )
```

```
; Точка ps0 определяется функцией переноса фрейма в DWG-файл.
(setq x (car  ps0)
      y (cadr ps0)
)
(if (= img "D")
 ;---------------------------------------------------------------------
 (progn ; Трехмерная графика.
  (cond
   ((member det '("2" "3")) ; Упрощенный вариант отрисовки.
    (@ace_lcl img_l nil)
    (@srtm_dwt2 x y b h s u tp) ; Рисую сечение.
    (setq ls (cons (entlast) ls))
    ; Дополняю во все примитивы группу 39 - высота.
    (mapcar '(lambda (x)
               (entmod (append (entget x) (list (cons 39 l))))
             )
             ls
    )
   )
   ((member det '("1"))  ; Максимальная детализация.
    (@ace_lcl img_l nil)
    (@srtm_dwt1 x y b h s u tp r1 r) ; Рисую сечение.
    (setq ls (cons (entlast) ls))
    ; Дополняю во все примитивы группу 39 - высота.
    (mapcar '(lambda (x)
               (entmod (append (entget x) (list (cons 39 l))))
             )
             ls
    )
   )
   (t (setq kodw '#break#)) ; Код возврата - отказ.
  )
 )
 ;---------------------------------------------------------------------
 (progn ; Плоская графика.
  (cond
   ((= det "3") ; Самое простое изображение.
    (cond
     ((member img '("1" "1F" "2" "2F"))
      (@ace_lcl img_l nil)
      (command "плиния" (list x y) "ш" 0 0 (list x (+ y l)) "")
      (setq ls (cons (entlast) ls))
     )
     ((member img '("3" "3F" "31" "31F"))
      (@ace_lcl img_l nil)
      (command "плиния"
               (list (+ x b) (+ y h)) "ш" 0 0 (list (- x b) (+ y h)) "")
      (setq ls (cons (entlast) ls))
      (command "плиния" (list x (+ y h)) (list x (- y h)) "")
      (setq ls (cons (entlast) ls))
      (command "плиния" (list (- x b) (- y h)) (list (+ x b) (- y h)) "")
      (setq ls (cons (entlast) ls))
     )
     (t (setq kodw '#break#)) ; Код возврата - отказ.
    )
   )
   ((= det "2")       ; Упрощенный вариант (без скруглений).
    (cond
     ((member img '("1" "1F"))
      (@ace_lcl img_l nil)
      (command "плиния" (list (- x b) y) "ш" 0 0 (list (- x b) (+ y l)) "")
      (setq ls (cons (entlast) ls))
      (command "плиния" (list (+ x b) y) (list (+ x b) (+ y l)) "")
      (setq ls (cons (entlast) ls))
      ; Торцы сплошной линией
      (command "плиния" (list (- x b) y) (list (+ x b) y) "")
      (setq ls (cons (entlast) ls))
      (command "плиния" (list (- x b) (+ y l)) (list (+ x b) (+ y l)) "")
      (setq ls (cons (entlast) ls))
     )
```

```
    ((member img '("2" "2F"))
     (@ace_lcl img_l nil)
     (command "плиния" (list (- x h) y) "ш" 0 0 (list (- x h) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     (command "плиния" (list (+ x h) y) (list (+ x h) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     ; Торцы сплошной линией
     (command "плиния" (list (- x h) y) (list (+ x h) y) "")
     (setq ls (cons (entlast) ls))
     (command "плиния" (list (- x h) (+ y l)) (list (+ x h) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     )
    ((member img '("3" "3F" "31" "31F"))
     (@ace_lcl img_l nil)
     (@srtm_dwt2 x y b h s u tp)
     (setq ls (cons (entlast) ls))
     )
    (t (setq kodw '#break#)) ; Код возврата - отказ.
   )
  )
  ; Максимальная детализация
  (t
   (cond
    ((member img '("1" "1F"))
     (@ace_lcl img_l nil)
     (command "плиния" (list (- x b) y) "ш" 0 0 (list (- x b) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     (command "плиния" (list (+ x b) y) (list (+ x b) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     ; Торцы сплошной линией
     (command "плиния" (list (- x b) y) (list (+ x b) y) "")
     (setq ls (cons (entlast) ls))
     (command "плиния" (list (- x b) (+ y l)) (list (+ x b) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     (@ace_lcl img_h nil)
     (command "плиния" (list (- x s) y) (list (- x s) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     (command "плиния" (list (+ x s) y) (list (+ x s) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     )
    ((member img '("2" "2F"))
     (@ace_lcl img_l nil)
     (command "плиния" (list (- x h) y) "ш" 0 0 (list (- x h) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     (command "плиния"
            (list (+ x (- h) tp) y) (list (+ x (- h) tp) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     (command "плиния" (list (+ x h) y) (list (+ x h) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     (command "плиния"
            (list (+ x h (- tp)) y) (list (+ x h (- tp)) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     ; Торцы сплошной линией
     (command "плиния" (list (- x h) y) (list (+ x h) y) "")
     (setq ls (cons (entlast) ls))
     (command "плиния" (list (- x h) (+ y l)) (list (+ x h) (+ y l)) "")
     (setq ls (cons (entlast) ls))
     )
    ((member img '("3" "3F" "31" "31F"))
     (@ace_lcl img_l nil)
     (@srtm_dwt1 x y b h s u tp r1 r)
     (setq ls (cons (entlast) ls))
     )
    (t (setq kodw '#break#)) ; Код возврата - отказ.
   )
  )
 )
);progn плоская графика.
;-------------------------------------------------------------------------
)
```

```
     (if (not (= kodw '#break#))
       (progn
         ; Создаю блок в масштабе 1:1.
         (command "БЛОК" (cadr nm_img) ps0)
         (apply 'command ls)
         (command "")
         (command "ВСТАВЬ" (cadr nm_img) ps0 1.0 1.0 0.0)
       )
     )
    )
    (progn
     ; Блок-изображение есть! Необходимо его только вставить.
     (command "ВСТАВЬ" (cadr nm_img) ps0 1.0 1.0 0.0)
    )
   );if
   (if kodw kodw '#ok#) ; Код возврата. Если не отказ, то все хорошо.
  );progn
 );if
)
;*****************************************************************************
'

;*****************************************************************************
; Функция отрисовки поперечного сечения двутавра с полной детализацией.
(defun @srtm_dwt1 (x y b h s u tp r1 r / a t45 c1 c2 c3 c4 c5 c6)
  (setq a   (atan u)   ; Угол alfa
        t45 (/ (sin (- (* pi 0.25) (* a 0.5)))
               (cos (- (* pi 0.25) (* a 0.5)))
            )
        c1  (- tp (* r1 t45) (* (/ (- b s) 2) u))
        c2  (* r1 (cos a))
        c3  (* r1 (- 1 (sin a)))
        c4  (* r t45 (cos a))
        c5  (- (* (- (* b 0.5) (* s 1.5)) u) (* r t45 (sin a)))
        c6  (+ (* r t45) (* (- (* b 0.5) (* s 1.5)) u))
  )
  (command "плиния"
    (list (+ x b) (+ y h (- c1)))
    "ш" 0 0
    (list (+ x b) (+ y h))
    (list (- x b) (+ y h))
    (list (- x b) (+ y h (- c1)))
    "дуга" "радиус" r1
    (list (+ x (- b) c3) (+ y h (- c1) (- c2)))
    "отрезок"
    (list (- x s c4) (+ y h (- tp) (- c5)))
    "дуга"
    (list (- x s) (+ y h (- tp) (- c6)))
    "отрезок"
    (list (- x s) (+ y (- h) tp c6))
    "дуга"
    (list (- x s c4) (+ y (- h) tp c5))
    "отрезок"
    (list (+ x (- b) c3) (+ y (- h) c1 c2))
    "дуга"
    (list (- x b) (+ y (- h) c1))
    "отрезок"
    (list (- x b) (- y h))
    (list (+ x b) (- y h))
    (list (+ x b) (+ y (- h) c1))
    "дуга"
    (list (+ x b (- c3)) (+ y (- h) c1 c2))
    "отрезок"
    (list (+ x s c4) (+ y (- h) tp c5))
    "дуга"
    (list (+ x s) (+ y (- h) tp c6))
    "отрезок"
    (list (+ x s) (+ y h (- tp) (- c6)))
    "дуга"
    (list (+ x s c4) (+ y h (- tp) (- c5)))
    "отрезок"
```

```
    (list (+ x b (- c3)) (+ y h (- c1) (- c2)))
    "дуга"
    (list (+ x b) (+ y h (- c1)))
    ""
  )
)
;*************************************************************************
;
;*************************************************************************
; Функция отрисовки поперечного сечения двутавра без скруглений.
(defun @srtm_dwt2 (x y b h s u tp / c0)
  (setq c0 (* (- b s) u 0.5))
  (command "плиния"
    (list (+ x b) (+ y h))
    "ш" 0 0
    (list (- x b) (+ y h))
    (list (- x b) (+ y h (- tp) c0))
    (list (- x s) (+ y h (- tp) (- c0)))
    (list (- x s) (+ y (- h) tp c0))
    (list (- x b) (+ y (- h) tp (- c0)))
    (list (- x b) (- y h))
    (list (+ x b) (- y h))
    (list (+ x b) (+ y (- h) tp (- c0)))
    (list (+ x s) (+ y (- h) tp c0))
    (list (+ x s) (+ y h (- tp) (- c0)))
    (list (+ x b) (+ y h (- tp) c0))
    (list (+ x b) (+ y h))
    ""
  )
)
;*************************************************************************
;
```

Выше в тексте функции отрисовки используется функция **@ACE_LCL**. Это универсальная функция для настройки слоя, цвета и типа линии для отрисовываемых объектов. Может (должна) использоваться во всех функциях отрисовки для гибкой настройки на необходимые слой, цвет и тип линии через описание в базе на уровне СРЕДЫ, т.е. объекты конкретной среды, входящей в состав ACE-системы, могут отрисовываться различным образом...

Функция имеет два аргумента. Оба аргумента - списки одинаковой структуры :

(*<слой>* - строковая, символ или nil
<цвет> - целое, символ или nil
<тип линии> - строковая, символ или nil
<имя файла для типа линии> - строковая, символ или nil
<путь для поиска файла с описанием линий> - строковая, символ или nil
)

Если имя слоя указано, то *<цвет>* и *<тип линии>* используются для настройки этого слоя, ели он не создан. Если этот слой уже существует, то этот слой устанавливается текущим, а *<цвет>* и *<тип линии>* не учитываются. Если *<слой>* не указан, но указаны *<цвет>* либо *<тип линии>,* то выполняется настройка на указанные *<цвет>* и *<тип линии>* для текущего слоя (режимы создания примитивов).

Первый аргумент список желаемых значений (например из базы).

Примеры:

```
("a" 2 "hidden")
("a" 2)
(nil nil "hidden")
nil
```

Второй аргумент список значения при отсутствии первого аргумента или его неполного объема.

примеры:

```
("s" 3 "dot")
(nil nil "dot")
nil
```

Если линия описана не в файле *acad.lin*, то следует указать имя файла и путь поиска, например : *("a" 3 "osi" "ace" ace_pt),* где "osi" - имя типа линии, "ace" - имя файла, ace_pt - путь поиска файла. Если путь не указан, поиск указанного файла выполняется в доступных директориях. Если не указан и файл, выполняется поиск файла *acad.lin* в доступных директориях. Во всех случаях 'цепочка' тип_линии-имя_файла-путь наследуется полностью, т.е. если в первом списке не указан путь, а во втором списке путь указан, все равно поиск будет выполняться по доступным директориям, а не по пути из списка второго аргумента.

Приложения.

Приложение А. Служебные слоты.

Имя слота	Тип	Краткое описание	Пример
ORDER_UP	Список	Порядок следования указателей наследования	(ORDER_UP UP6 UP2 UP UP4)
ACE_UP	Список	Список определенных указателей наследования	(ACE_UP UP UP2 UP4 UP6 ...)
ADPT	Символ	Флаг включения адаптивной базы	(ADPT . T) ; адаптивная база включена, см. файл ace_set.fr
APP	Список	Список сливаемых ключей	(APP KEYF APP_KEY KEY_DEMON)
APP_KEY	“Разворачиваемый” слот	Список отображаемых монитором характеристик	(APP_KEY & (H B Jx Jy ...))
BEFORE	Демон	Демон предобработки фрейма	(before # () (progn (if (not @fr_ld) (load_1 ACE_MAIN_pt "fun\\@fr_ld.bi4")) (@fr_ld _#frm)))
CUT	Выражение	предикат 0-го элемента для усечения семейства	(CUT # (cut_0) cut_0), где cut_0 может иметь следующий вид: (CUT_0 # (sokr #select#) (cond ((and (get-val 'prf_sokr ACE_SET) (not sokr)) nil) (#select# (= #select# '#ok#)) (t t)))
DET	Число/ строка	Степень детализации	
EXT_NM	Символ	Внешнее имя	(EXT_NM . SRTM_DWTR)
FIND	“Вычисляемый” слот	Слот организации поиска фрейма	(find . (cond ((/= (type ACE_NM_TEST) 'STR) nil) (t (list (substr ACE_NM_TEST 1 4)))))
FML	А-список	Семейство фреймов	(FML (; 0-ой элемент (UP ((NM . “Нормальные двутавры”) (FR DWTB “DWTB.FR”) ...) (BEFORE <слот предобработки>) (CUT <слот усечения>) ...) ((NM . “ДВТБ_10Б1”) ...) ;1-ый фрейм ((NM . “ДВТБ_12Б1”) ...) ;2-ой ((NM . “ДВТБ_12Б2”) ...) ;3-ий ... ((NM . “ДВТБ_100Б4”) ...) ;i-ый)
IMG			
KEY_DEMON	“Разворачиваемый” слот	Список демонов фрейма	(key_demon & (move hlp_sld hlp_txt hlp_pic hlp_flm dem_print))

Имя слота	Тип	Краткое описание	Пример
KEY_HLP	“Вычисляемый” слот	Список отображаемых монитором характеристик	(KEY_HLP # (APP_KEY) (APP_KEY))
KEY_SPRT	Список	Список служебных ключей ACE-технологии	(key_sprt fml ace_up adpt app before after key_demon cut find keyf key_sprt key_hlp app_key histr dostup)
KEYF	А-список	Атрибуты ключа	см. приложение Б.
LD_AKK	Символ	Имя списка-аккамулятора дозагрузки	(ld_akk . SRTM_PRF_AKK)
LD_FILE	Строка	Имя файла дозагрузки	(ld_file . "bz0/dwtb_.fr")
LNG	А-список	Список языков диалога	(LNG ("RU" . "Русский") ("EN" . "Английский") ("UA" . "Украинский") ...)
MIR	Строка	Признак зеркаленья	
NM	Строка	Внутреннее имя фрейма	(NM . “DWTR_20”)
SCL	Число	Масштаб	
SI	А-список	Список систем единиц измерения	(SI ("СИ" . nil) ("Техн" . "si/ace_si") ("Техн1" . "si/ace_si1") ("Англ" . "si/ace_si2") ...)
TYK			
TYP_OBJ	Строка	Тип объекта	(TYP_OBJ . “PRF”)
UP	А-список	Наследование по умолчанию	(UP (NM . “DWTR”)(FR SRTM_DWTR “DWTR0.FR”) (PT . SRTM_PATH))
UP2, UP3, ...	А-список	Дополнительные указатели наследования	(UP4 (nm . "DWTR_20") (fr find (nm . "*SRTM*")) (keyf (nm (txt . "Двутавр 20") (lng Rdwtr "SRTM"))))

Приложение Б. Структура слота KEYF.

Ключ	Тип значения	Краткое описание	Примеры
FLM	список	фильм-справка о ключе	(FLM (("all.flc")) injn_img_pt)
HLP	список	слайд-справка о ключе	(HLP "bz0/sld/injn0" INJN_PATH)
HLT	список	текстовая справка о ключе	(HLT ("bz0/txt/hlp_??.hlp") INJN_PATH)
LNG	строка/ список	словарь и правила перевода языка диалога	(LNG . "INJN")
MOD	список	выражение модифицирующее ключ	(MOD (@send '((nh . 1)) ACE_FR_NM nil))
PIC	список	картинка-справка о ключе	(PIC (("all.jpg")) injn_img_pt)
RUN	список	выражение для выполнения Указания	(RUN (@send '((draw . 1)(nh . 0)) ACE_FR_NM nil))
TXT	строка	текстовое наименование ключа	(TXT . "Арматура")
TYP	список	список для монитора	(TYP view exit exit_mod)
UNT	строка	размерность значения ключа	(UNT . "мм")

Приложение В. Глобальные переменные.

Имя переменной	Описание
ACE_PATH	Путь к ACE-технологии.
ACE_MAIN_PT	Путь к главному каталогу ACE-технологии.
SRTM_PATH	Путь к файлам сортамента металлопроката.
ARCH_PATH	Путь к файлам архитектурных элементов.
LIGT_PATH	Путь к файлам
INJN_PATH	Путь к файлам
STNK_PATH	Путь к файлам
ARCH_BLD_PATH	Путь к файлам архитектурно-строительной среды.
ELECTR_PATH	Путь к файлам электротехнической среды.
GEO_PATH	Путь к файлам генлана.
SAN_TECH_PATH	Путь к файлам сантехнической среды.
TECHNL_PATH	Путь к файлам технологической среды.
SRTM_IMG_PT	Путь к файлам растровых изображений и анимации сортамента металлопроката.
INJN_IMG_PT	Путь к файлам растровых изображений и анимации инженерного оборудования.
STNK_IMG_PT	Путь к файлам растровых изображений и анимации станочного оборудования.
@SEND_ALERT	Если эта переменная не nil, то функция @SEND генерирует сообщение о своей работе.
@FIND_FR_ALERT	Если эта переменная не nil, то функция @FIND_FR генерирует сообщение о своей работе.
#GSLOTS_ALERT	Если эта переменная не nil и режим #@CTRL не 2, то функция #Gslots генерирует предупреждения, о неопределенных аргументах при вычислении слотов-выражений.
MAX-ELEMENT-CACHE-OF-STRINGS	Максимальное число элементов кэша строк. В ACE_PATH.SET. По умолчанию - 400.

Приложение Г. Служебные переменные.

Имя	Место определения	Краткое описание	Примеры
_#FRM	вычисляемый слот / демон	текущий фрейм для которого вызывалась функция @Get-Slots.	
_#OLD			

Приложение Д. Коды возвратов демонов и вычисляемых слотов.

Код	Описание
#ok#	Признак успешного окончания работы демона.
#break#	Признак прерывания вычисления демона (передается по всей цепочке демонов и/или вычисляемых слотов)
(#break# ...)	
#repeat#	Признак необходимости повторить вычисления демона (передается по всей цепочке демонов и/или вычисляемых слотов)
(#repeat# ...)	Признак
(#interrupt# ...)	

Приложение Е. Зарезервированные символы.

Символ	Описание
Any:	Признак неконтролируемых аргументов. Список аргументов вычисляемых слотов и демонов.
Need:	Признак недостающих характеристик. Список аргументов вычисляемых слотов и демонов.

Список служебных слотов.

NM - Единственный обязательный слот в систеиме. Внутреннее имя фрейма. Значение этого слота используется для идентификации фрейма в системе и должно быть уникально в ее пределах. Значением данного ключа может быть либо строка, либо список строк.

UP?? - Слоты наследования. Это служебный слот, имеющий сложную структуру. Существует несколько вариантов организации данного слота - путем непосредственного указания родительского фрейма и путем задания внутреннего имени фрейма и признака поиска:

ВАРИАНТ 1:

```
(up
  (nm . "DWTB0")
  (fr DWTB0 "BZ/DWTB0.FR")
  (pt . SRTM_dwtb)
)
```

ВАРИАНТ 2:

```
(up
  (nm . "DWTB0")
  (fr FIND (nm . "*SRTM*"))
)
```

В обеих случаях подслот NM является обязательным, так как указывает на имя родительского фрейма. Первый вариант можно использовать если известно внешнее имя фрейма, имя файла содержащего этот фрейм и путь доступа к указанному файлу. Путь поиска - строка, указывающая либо полный путь поиска, либо относительный. В том случае когда не известно внешнее имя фрейма ссылка на него осуществляется с помощью признака поиска, при этом неоходимо указать имя среды в которой будет осуществляться поиск.

Первый вариант более быстрый, используется при последовательном движении по пути наследования. Второй - требует меньше исходных данных, используется при ссылках из семейства FML на какие-либо среды не на текущем пути наследования.

FML - семейство фреймов, содержащихся в данном фрейме, реализующих как отношение ЯВЛЯЕТСЯ , так и отношение СОДЕРЖИТ. 0-й элемент семейства системой ВСЕГДА рассматривается как общая часть информации по всему семейству и система каждый элемент семейства, начиная с 1-го воспринимает как слитый с 0-м элементом В 0-й элементе могут находитьсяч слоты :

CUT - выражение-тест для усечения-сортировки элементов семейства FML

FIND - выражение для поиска-предобработки имени элемента семейства, возвращает nil, либо список вида

'("ЛСТГ_20" (h . 120.0) (nm . "ЛСТГ_20x120").....)

соглашения по ключу FIND следующие :

- если ключа нет, поиск идет по заданному для поиска слоту NM
- если ключ есть и его значение NIL прекращаем как неудачный
- если значение список, первый элемент используем для поиска элемента семейства, а хвост (а-список) логически сливаем с найденным элементом (ранее объединенным с 0-м)
- внутри FIND для тестирования может использовать, по крайней мере переменную ACE_nm_test

FR - Ссылка на фрейм. Этот слот может быть использован для организации как ссылок вниз (на семейства), так и ссылок вверх (организация наследования с помощью UP). На 14/12/95 слот имеет два варианта формата:

ВАРИАНТ_1 - (FR DWTR0 "BZ/DWTR0.FR"), где DWTB0 - символ-имя фрейма на который Вы хотите сослаться; "BZ/DWTR0.FR" - имя файла, который должен быть загружен в том случае, когда указанный слот не найден или не прошел проверку на соответствие по слоту NM (см. ниже). При загрузке файла может быть использована информация о пути к файлу (слот PT), если он присутствует.

ВАРИАНТ_2 - (FR FIND (NM . "*SRTM*")), где FIND - признак поиска, указывающий, что система должна осуществить поиск фрейма; (NM . "*SRTM*") - указатель среды в которой будет осуществляться поиск.

В обеих вариантах в качестве первого слота ДОЛЖЕН присутствовать слот NM. В ПЕРВОМ варианте он используется при проверке фрейма на соответствие (на случай возвожного замещения одного фрейма другим, либо иного изменения оригинального содержимого). Если система обнаружит несоответсвии информации из слота NM и из указанного фрейма, либо если указанный фрейм вовсе не существует, то выполнится загрузка указанного файла. Во ВТОРОМ варианте слот NM задает признак поиска фрейма.

PT - Вспомомогательный слот. Необходим при задании слота FR с указанием загружаемого файла (вариант 1).

BEFORE - Демон предобработки фрейма. Вычисляемое выражение специального типа, выполняющееся до основной обработки фрейма.

AFTER - Демон постобработки фрейма. Вычисляемое выражение специального типа, выполняющееся после основной обработки фрейма.

APP - Список слотов, значения которых при наследовании не замещаются а сливаются, например (app keyf key_demon app_key).

KEY_DEMON - Список демонов фрейма. Используется с учетом слияния по уровням за счет механизма ключа APP

KEY_HLP - Список ключей выводимых монитором в группу ХАРАКТЕРИЗУЕТСЯ:, а также гарантированно выводимых функцией поиска @find_fr при управляющем параметре '(... (fr . t) ...). Вычисляемый через 'app_key

APP_KEY - собираемые по уровням ключи в key_hlp

Приложение Ж. Локальные переменные монитора.

Переменная	Описание
ACE_fr_nm	Имя текущего фрейма основного
ACE_fr_nmi	Имя текущего фрейма i-го дочернего
ACE_fr_vl	Значение текущего фрейма
ACE_fr_00	Копия ACE_fr_nm при ACE_fr_nmi
ACE_RUN_MOD	Флаг - Возмож модиф фрейма демоном
ACE_SEND_RUN	Рабочее выражение
ACE_RUNNH	Выражение - Настройки, модификация
_#cur	Текущий фрейм не имеющий внешнего имени
lmt_up	Список ограничения наследований
#nmprj	Имя верхнего проектного фрейма для синхронизации
@_set_dem	Текущее имя демона 'hlp_dem / 'drawing / "Указания :"
ace_climg	Цвет фона
@_00ls	Отобранная строка-список СУПЕРСПИСКА
@_dcl_sel	Предшествующие ключи окна
@_00l	Суперсписок основной
@_dml	Суперсписок указаний
@_dmli	Суперсписок указаний дочерний
@_upl	Список отобранных демонов
@_up0	Служебн список для "Выполнено из"
@_fml0	Семейство начальное
@_sld_000	Инф по группе слайдов
@_sld	Текущ инф по группе слайдов
dcl_00	
dcl_00nm	
done	
tmp	
@_ln	Язык
fl_pse	Флаг паузы
fl_term	Флаг повторного входа в окно
slct	Флаг справки / выбора фрейма
sel_fli	Флаг выбора из множ, для i-го
sel_fl	Флаг выбора из множ, основн
#kf#	Изменения keyf на сеанс выбора из множества
#hl#	Изменения любых ключей на сеанс выбора из множества, особенно 'key_hlp
fl_active	Флаг уже открытого окна
z_txt	Текст для справки и выбора из множ
s_txt	Текст для текущего объекта