# Глава 6 Создание сценариев отладчика

Отладка вдвое сложнее, чем написание кода. Поэтому -

если вы пишете код настолько умно, насколько это возможно, вы, по определению,

недостаточно умны для его отладки.

-Брайан В. Керниган и П. Дж. Плаугер,

Элементы стиля программирования

Я ненавижу жуков, будь то жуки в моих программах или те, у которых пятьсот ног и с двумя ярко-красными клещами, которые ползают по моей ноге, когда я их не вижу, и которые довольно распространены в Австралии.

Чтобы избежать ошибок, в главе 4 вы узнали, как можно применять контракты в программах, чтобы выявлять фатальные ошибки на ранних стадиях и выполнять динамические тесты как неотъемлемую часть написания программ с помощью разработки на основе тестирования. Конечно, также можно использовать ABAP Test Cockpit для выполнения статических проверок вашего кода. Эти элементы представляют собой трехстороннюю атаку на ошибки, пытаясь уничтожить их до того, как они пустят корни.

Однако ошибки всегда с нами; программа почти никогда не делает всего того, что от нее хотели, без ошибок при первом запуске в процессе разработки или QA. (Это несмотря на то, что Роберт К. Мартин в своей книге "Чистый кодер" настаивает на том, что "QA не должен ничего находить"). Вы всегда должны ожидать, что первые несколько тестов - своих или других - дадут неожиданные результаты.

Если кто-то другой скажет вам, что программа-симулятор монстра производит синих монстров с двумя головами, тогда как вы ожидали зеленых монстров с тремя головами, тогда вам нужно определить источник проблемы. Если вы когда-нибудь делали это - а я подозреваю, что делали - то вы знаете, что бывают дни, когда вы проводите четыре часа в поисках проблемы, а когда она, наконец, появляется, решение становится настолько очевидным, что вы можете внести изменения в код, чтобы устранить проблему за две минуты.

281

Проблема в том, что вы сидите перед отладчиком и ищете одно и то же в разных программах, снова и снова, месяц за месяцем, год за годом, когда лучше бы вы потратили это время на написание программ. Что если бы вы могли решить эту проблему, занимаясь именно этим - написанием программ, превращая таким образом нечто действительно скучное занятие в нечто действительно увлекательное и быстрее решить проблему вашего работодателя в то же самое время? Тогда все были бы счастливы. Хорошие новости: вы можете! В этой главе мы расскажем, как это сделать. Чтобы провести вас через процесс написания скриптов отладчика, сначала вы увидите общий вид написания программы сценария отладчика - с использованием вкладки Script, которая является вашей отправной точкой (раздел 6.1). После этого вы узнаете о технических деталях кодирования метода SCRIPT (раздел 6.2), в котором происходит 99% действий. В разделе 6.3 приводится пример, показывающий, как могут помочь друзья метода SCRIPT - INIT и END.

Наконец, вы увидите несколько примеров практического использования сценариев отладчика в реальной жизни (раздел 6.4).

Написание сценариев вне программы

На самом деле существует два способа создания сценария отладчика: во время отладки программы и совершенно независимо. Эти методы практически идентичны и в этой главе основное внимание уделяется объяснению процесса изнутри отлаживаемой программы. Однако, если у вас есть невероятно сложный сценарий отладчика, который вы не хотите писать его в процессе отладки программы, вы можете использовать SAS. Это позволит вам создать сценарий отладчика в автономном режиме. (Если вы живете в Великобритании, вы можете почувствовать нестерпимое желание крикнуть "Кто смеет, тот побеждает! при использовании этой транзакции.)

Эта транзакция выглядит более или менее точно так же, как и те, что обсуждаются в этом разделе. Разница лишь в том, что список файлов трассировки находится у вас перед носом, а собственно инструмент для написания сценариев находится на третьей вкладке слева (вкладка Редактор сценариев).

## 6.1 Написание программы сценария отладчика

Поскольку вы читаете книгу под названием ABAP to the Future, мы надеемся, что вы уже установили настройки отладчика таким образом, что находитесь в "новом" отладчике (который на самом деле уже довольно старый). Чтобы проверить это, зайдите в Transaction SE80, выберите пункт меню *Utilities - Settings*, и перейдите на вкладку ABAP Editor. Здесь вы увидите еще один набор меньших вкладок под основной вкладкой ABAP Editor. Нужная вам вкладка - Отладка. Здесь вы можете выбрать между Классическим (SAP говорит о старом/мусорном) и Новым, выбрав

282

соответствующую радиокнопку. Вам нужно выбрать New, чтобы пройти через примеры в этой главы. После того как вы выбрали новый отладчик, при отладке вы увидите полосу вкладок в верхней части страницы, показывающую различные представления отладчика, которые вы можете просматривать (Рисунок 6.1).



Figure 6.1 Debugger Tab Strip

Как и следовало ожидать, вкладка Script позволяет создавать сценарии отладчика. Перейдя на эту вкладку, вы попадаете на экран, который выглядит так, как показано на рисунке 6.2.

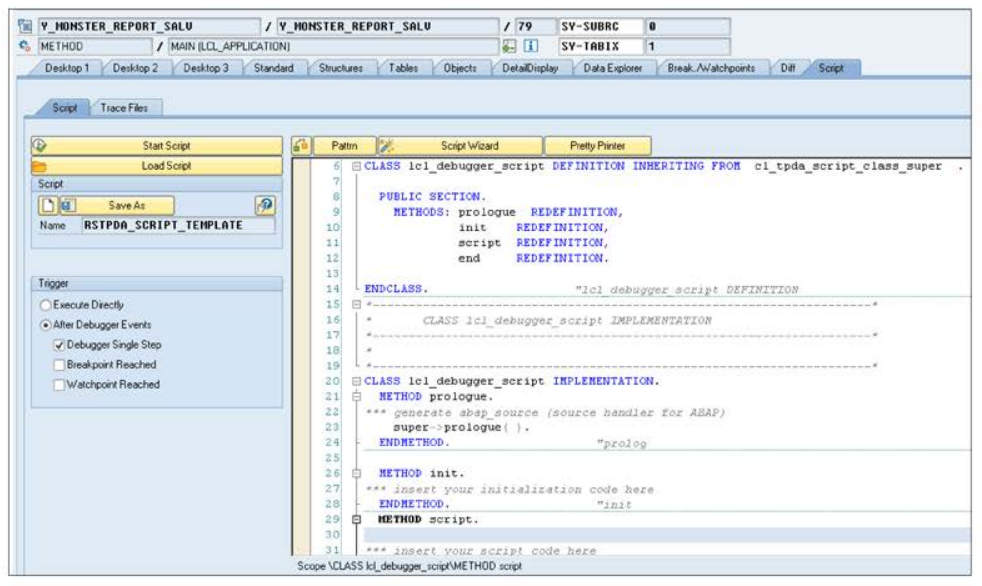


Figure 6.2 Debugger Script Blank Template

Примечание

Если вы перейдете на вкладку Сценарий в любой системе, отличной от клиента разработки, в котором вы создаете свои программы, вы увидите сообщение System Set to Not Changeable message. Это происходит потому, что сценарии отладчика сами являются программами, и это, вероятно, является

283

плохой идеей писать какую-либо программу непосредственно в (скажем) производстве. Вы можете загружать существующие скрипты из базы данных в отладчик в продакшене, однако, так что это не является проблемой в каком-либо реальном смысле.

Кто-то может сказать: "Погодите, мне нужно запускать программу в среде с данными, поэтому я никогда не отлаживаю программу в процессе разработки". Ответ на это заключается в том, что вы забываете о модульных тестах; они выполняются в процессе разработки и, по своей природе, должны охватывать 100% кода.

Часть экрана справа, показанная на рисунке 6.2, представляет собой не что иное, как обычный редактор ABAP - что означает, что весь спектр команд ABAP находится в вашем распоряжении.

В верхней части экрана, в разделе PUBLIC SECTION, вы можете увидеть, что есть четыре метода, которые вы можете использовать при написании сценариев отладчика. Один из них не должен переопределяться. Два других могут быть переопределены, но не обязательно. Четвертый должен быть переопределен. Метооды выглядят следующим образом:

1. Метод ***PROLOGUE*** (не переопределять)

Метод PROLOGUE запускается один раз в начале сценария и выполняет некоторую жуткую черную магию, чтобы убедиться, что остальная часть вашего сценария отладчика может получить доступ к переменным в отлаживаемой программе. Вы не добавляете никакого своего кода в этот метод; он выполняется автоматически, без вашего участия.

2. Метод ***INIT*** (может быть переопределен)

Метод INIT запускается один раз в начале сценария и чем-то похож на метод SETUP в модульном тесте. Он позволяет установить некоторые переменные в их начальные значения (например, создать экземпляр приложения Jog).

3. Метод ***END*** (может быть переопределен)

Метод END похож на метод *TEARDOWN* в модульном тесте. Он запускается один раз в конце сценария и позволяет вам сделать такие вещи, как сохранить журнал приложения в базу данных или отправить себе результаты по электронной почте. Важно отметить, что этот метод будет запущен только в том случае, если программист вручную нажмет кнопку Exit Script. (вы увидите, как это сделать в разделе 6.3.)

4. Метод ***SCRIPT*** (должен быть переопределен)

Метод SCRIPT - самый важный метод; он содержит логику внутри программы, которая будет использоваться для улучшения процесса отладки. Этот метод запускается по крайней мере один раз, но, скорее всего, много раз - обычно каждый раз, когда вы проходите через строку в коде, который вы отлаживаете.

284

Другим элементом, показанным на рисунке 6.2, является возможность настройки триггеров; эти настройки более подробно показаны на рисунке 6.3. В левой части экрана есть две радиокнопки: Выполнять непосредственно и После События отладчика. Если вы выберете первый вариант, а затем выполните свой сценарий естественным образом, то он будет выполняться прямо там и тогда, используя информацию, доступную в тот момент, когда вы находитесь в отладчике.



Figure 6.3 Trigger Settings

Если вы выбрали "После событий отладчика" и затем выбрали "Выполнить", то вы контролируете, сколько раз сценарий будет выполняться, если вообще будет выполняться, на основе дальнейших настроек, которые вы делаете во флажках под кнопкой After Debugger Events:

- **Debugger Single Step**

Если вы выберете этот вариант, то ваш скрипт будет запускаться после выполнения каждого отдельного ABAP-оператора. Это нужно для тех случаев, когда вы совершенно не знаете, в каком месте программы происходит плохое событие, которое вы исследуете.

- **Breakpoint Reached**

Вы знаете, что такое точка останова, и это не отличается от нее. Каждый раз, когда указанная достигается точка останова, скрипт будет запущен. Это нужно для тех случаев, когда вы подозреваете, что проблема возникает после определенного оператора, например AUTHORITY\_CHECK, или после вызова определенного ФМ. Обратите внимание, что с появлением нового отладчика возможности точки останова были значительно расширены.

- **Watchpoint Reached**

И снова, точка наблюдения возникает при изменении значения указанной переменной, и каждый раз, когда это произойдет, ваш скрипт будет выполняться. Это нужно для тех случаев, когда вы знаете, что проблема вызвана тем, что переменная установлена с сомнительным значением, и вы хотите узнать, что именно ее вызывает.

Вам простительно думать, что ваш сценарий остановится в динамических точках прерывания и точках наблюдения, которые вы уже установили во время сеанса отладки, но вы ошибаетесь. На самом деле, точки останова и точки наблюдения специфичны для конкретной программы

285

и должны быть определены соответствующим образом. С этой целью, когда вы выбираете флажки "точка разрыва" или "точка наблюдения", вы увидите маленький значок карандаша, который появляется справа (Рисунок 6.4).

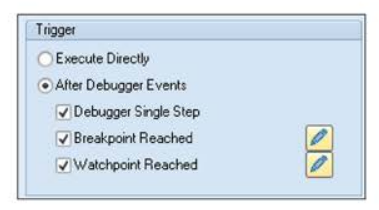


Figure 6.4 Breakpoint/Watchpoint Pencil Icons

Когда вы нажимаете на значок карандаша, появляется новый подэкран, в котором вы можете установить ваши точки останова и точки наблюдения точно так же, как и в обычном отладчике Рисунок 6.5 и Рисунок 6.6.

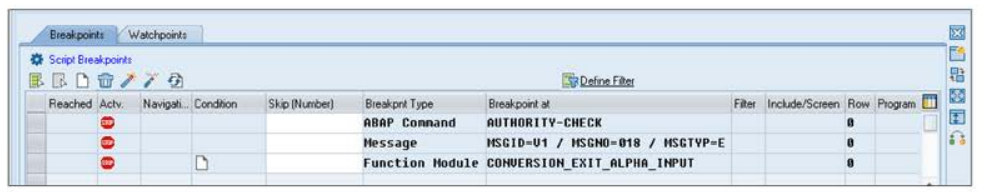


Figure 6.5 Script Breakpoints

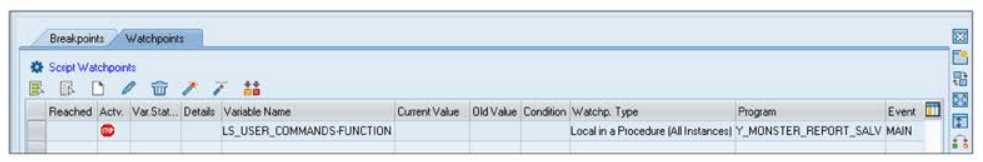


Figure 6.6 Script Watchpoints

Пока вы определяете точки останова и точки наблюдения, вы будете помнить о том, насколько гибки существующие опции в отношении того, где вы хотите, чтобы отладчик остановился. Однако ничто не бывает бесконечно гибким; однажды вы можете решить, что хотите останавливать отладчик только на определенном сообщении, когда определенная переменная имеет определенное значение, и есть определенная программа выше в текущем стеке вызовов, и PIO имеет заданное значение, и внешняя система находится в автономном режиме, и Юпитер находится в соединении с Плутоном, и петух прокричал четыре раза в полночь и снес четыре яйца. Это

286

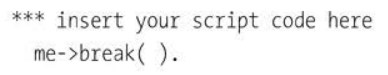
подобное требование является довольно распространенным, и вы можете удовлетворить его путем кодирования дополнительных проверок в начале метода вашего главного сценария, когда он вызывается точкой останова или точкой наблюдения.

## 6.2 Кодирование метода SCRIPT

Неизвестный человек однажды сказал: "Не антропоморфизируйте компьютеры, они это ненавидят". Тем не менее, в данном случае вам нужно подумать о том, как можно попросить компьютер вести себя так, как вы бы вели себя при отладке.

Для начала подумайте о том, как вы отлаживаете что-то сами. Во-первых, вы продолжаете через программу, пока не найдете то, что вас интересует. Для этого вы смотрите на экран. Ваш сценарий отладчика не может этого сделать, поэтому ему нужен способ доступа к переменным в запущенной программе, чтобы он мог анализировать их значения. Далее, вы можете изменить переменную или значение, пройти вперед по отлаживаемой программе на строку или решить, что вы находитесь не в том месте, и нажать кнопку F8, чтобы перейти к следующей точке останова или точке наблюдения.

Ваш сценарий отладчика должен уметь делать все эти вещи. Чтобы начать, перейдите к SCRIPT в вашей программе сценария отладчика. Там уже есть одна строка кода (листинг 6.1).



Listing 6.1 Generic Command

Это просто общая команда для прекращения выполнения вашего сценария и возврата управления

отладчику. Как вы можете себе представить, вы хотите добавить несколько строк кода перед этой командой. В верхней части раздела ABAP Editor вкладки Script отладчика вы увидите четыре кнопки. Три из них делают то, что вы ожидаете от редактора ABAP: проверка синтаксиса, "красивая печать" и вставка шаблона. Однако есть дополнительная кнопка, специфичная для отладочных сценариев: она называется Script Wizard (хотя у нее множественное расстройство личности); когда вы наводите на нее курсор, вы видите Script Services). Важно то, что на этой кнопке изображена волшебная палочка. Это должно быть хорошо!

Script Wizard - это замечательный волшебник, если вообще волшебник, потому что он служит мостом между отлаживаемой программой и программой, которая управляет отладчиком. Когда вы выбираете опцию в Script Wizard

287

некоторый код вставляется в вашу программу сценария отладчика точно таким же образом, что кнопка Pattern вставляет ФМ или вызов метода и заставляет вас заполнить сигнатуры (входные/выходные) значения.

На рисунке 6.7 показаны некоторые из наиболее интересных опций Мастера сценариев. Каждая из этих опций выполняет один из шагов, которые программист должен был бы выполнить вручную во время отладки программы.

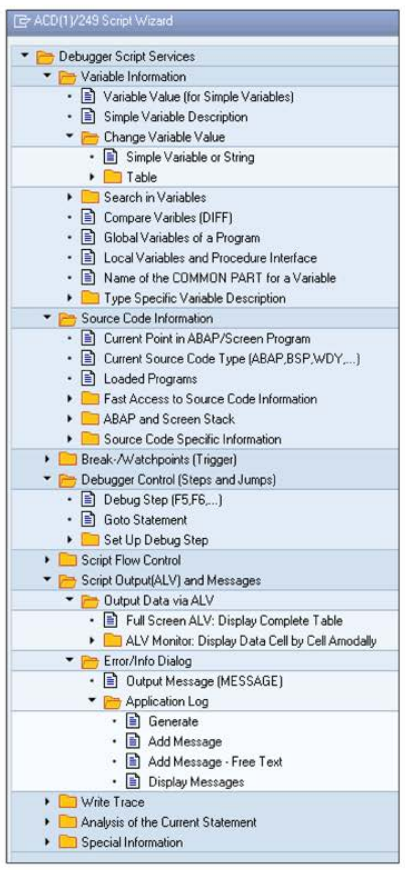


Figure 6.7 Script Wizard Options

288

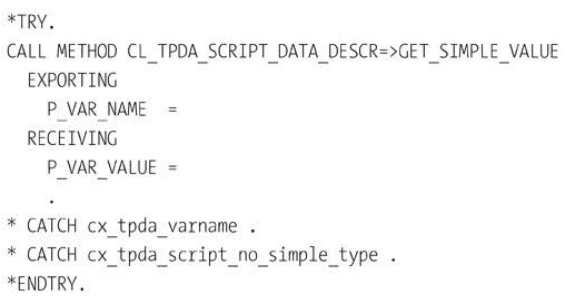
Например, некоторые опции предоставляют программе сценария значения различных переменных в отлаживаемой программt (Variable Information), информацию о системном стеке (ABAP и Screen Stack), или значения PIO (Special lnformation) - практически все, что вы можете увидеть, порывшись вручную в экране отладчика. Есть также несколько опций для управления шагом через отладчик для имитации нажатия, например, F5 или F8 (Управление отладчиком). Некоторые опции позволяют изменять значения переменных в отлаживаемой программе (Change Variable Value) - и позвольте мне сказать вам, что это намного проще добавить строку внутренней таблицы с помощью программы, чем заполнять все ячейки вручную.

Наконец, некоторые опции эквивалентны записи некоторой информации на вывод ALV, сообщения или запись в журнал приложения (вывод сценария (ALV) и сообщения). Это устраняет риск того, что используемая информация, которую вы написали на листе бумаги, потеряется, когда бухгалтер по основным средствам придет жаловаться на то, что ваша программа преобразования данных не работает, споткнется и прольет свой напиток на ваш стол.

Честно говоря, опции на этом экране должны вызывать у вас желание подпрыгнуть и кричать во весь голос: "Посмотрите на это! Посмотрите, как это хорошо!" Но более разумным наблюдением будет то, что над этим явно много думали, и кажется, что большинство, если не все, ручные действия во время отладки можно заменить в программе-скрипте отладчика. Давайте рассмотрим подробнее, так ли это.

Ручные действия программиста во время отладки можно разбить на четыре этапа: смотреть, двигаться, действовать и регистрировать.

Цель этапа поиска - найти значение одной из переменных в отлаживаемой программе. Для этого в мастере сценариев откройте его и перейдите в раздел на *Variable Information – Variable Value*. В код отлаживаемого сценария вставляется пустой шаблон ger script code, как показано в листинге 6.2.

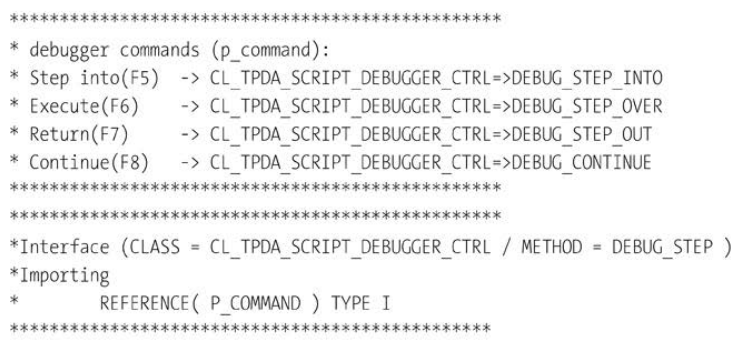


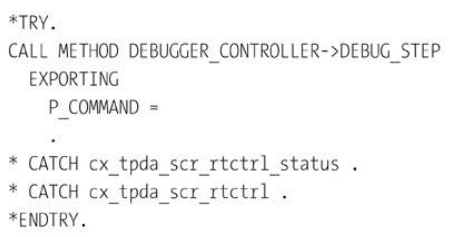
Listing 6.2 Variable Value

289

Идея заключается в том, что вы изменяете шаблон, чтобы передать в него имя переменной из отлаживаемой программы в виде строки с заглавными буквами, например, MONSTER\_TYPE. Вызовы метода перемещают значение переменной отлаживаемой программы в переменную, которая является локальной для вашей программы-скрипта отладчика. Затем возникает исключение, если вы передадите имя несуществующей переменной или, в данном конкретном примере, исключение если данная переменная не является элементарной переменной. Таким образом, вы программно, а не вручную, завершили этап поиска.

Теперь вы переходите к этапу перемещения. При отладке вручную вы перемещаетесь по отлаживаемой программе, пока не достигнете точки останова или точки наблюдения (которая находится расположенной в интересующей вас точке). И как только вы там окажетесь, вы, как правило, пройдете через несколько (или много) строк программы, нажимая F5 несколько раз. Эквивалентной опцией из Мастера сценариев является *Debugger Control - Debug Step*, которая генерирует код в листинге 6.3.





Listing 6.3 Debug Step

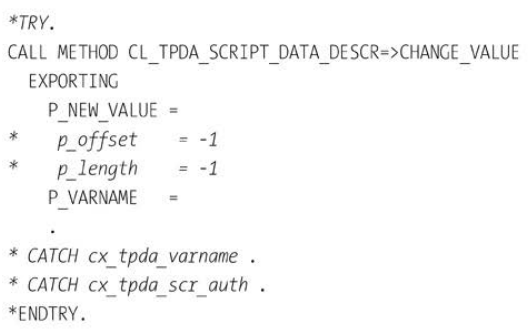
В сгенерированных комментариях перед методом DEBUG\_STEP вы увидите хороший список того, какие константы соответствуют каждой из функциональных клавиш в отладчике. Чтобы продвинуть

290

отладчик с помощью оператора, вы передаете константу DEBUG\_STEP\_INTO в метод DEBUG\_STEP. Тогда вы продвинетесь на одну команду дальше и можете вызвать другой метод, чтобы посмотреть на состояние системы и убедиться, что вы нашли ошибку. В данном примере, ошибка заключается в том, что наша переменная изменилась, и этот этап закончится тем, что вы найдете эту переменную.

Теперь, с помощью комбинации поиска и перемещения, вы нашли место, где находится ошибка и настало время для стадии действия. На этапе действия при ручной отладке вы обычно изменяете значение переменной на правильное, затем нажимаете F8, чтобы проверить, устранило ли изменение переменной проблему, так что остальная часть программы выполняется корректно. Если да, то вы знаете, что нужно сконцентрироваться на исправлении той части программы, которая испортила значение переменной.

Для этого в Мастере сценариев выберите *Change Variable Value – Simple Variable* или *Change Variable Value – Table*. Для данного примера используйте первый вариант; вы получите код, показанный в листинге 6.4.



Listing 6.4 Programatically Changing Varia ble Value in Debugger

Естественно, это просто обратный процесс чтения значения переменной, с тем же видом обработки ошибок в случае, если передано фальшивое значение переменной. После того, как вы изменили значение переменной, верните управление в обычный отладчик с помощью команды ME->BREAK().

Надеюсь, изменение значения переменной решило проблему. Если нет, то по крайней мере вы устранили одну возможную причину ошибки - но пришло время вернуться и вам придется изменить свой сценарий, чтобы найти что-то еще.

В любом случае, вернувшись в реальный отладчик, нажмите кнопку F8, что приведет к естественному завершению программы (и, следовательно, сеанса отладки).

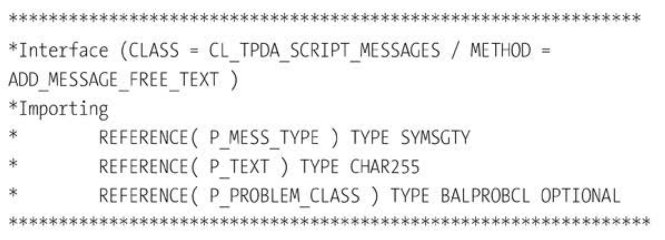
291

Этап регистрации происходит одновременно с тремя другими этапами. При прохождении отладки вручную, вы иногда делаете скриншоты значений некоторых переменных в определенный момент времени и распечатываете их, отправляете по электронной почте коллеге или записываете на бумаге. (Загрузка содержимого внутренней таблицы в электронную таблицу - еще одно забавное занятие). По сути, вы копируете информацию с экрана отладчика на более постоянный носитель.

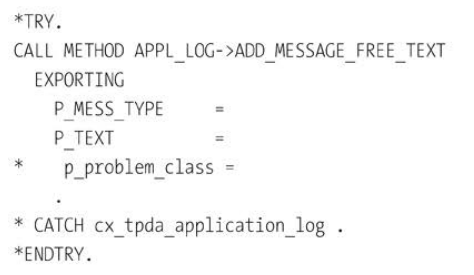
В качестве примера того, что вы можете захотеть записать в журнал, при просмотре экрана отладчика: при ручной отладке полезно посмотреть на стек вызовов, чтобы выяснить, как именно вы получили процедуру, которая сама себя неправильно ведет. Например, я вспоминаю, как прыгал от радости, когда я наконец нашел точную последовательность вызовов, используемую BAPI для создания заказа на продажу, чтобы добраться до кода, который устанавливает характеристики конфигурации варианта. В течение многих лет я не мог понять, где это происходит, а потом в один прекрасный день ("к счастью") BAPI начал сбрасывать данные именно в том месте, которое я искал. Я записал последовательность вызовов на листе бумаги и пометил этот лист бумаги как очень важный и подлежащий хранению в надежном месте. Потом, конечно, я его потерял.

К счастью, при использовании программы-скрипта с отладчиком нет необходимости проходить через всю программу и записывать значения переменных или состояние системы (то есть, какая именно процедура в какой именно программе изменила значение). Идея заключается в том, что ваша программа следит за движением отлаживаемой программы, собирает всю необходимую информацию, которая вам нужна, и затем показывает вам конечный результат за один раз. Отладчик предназначен для создания файлов трассировки, но вы также можете иметь ALV отчет, журнал приложения или даже электронную таблицу значений из внутренней таблицы по электронной почте, если вы напишете необходимый для этого код.

Однако если в сценарии отладчика есть момент, в котором вы хотите сделать запись о чем-либо, просто нажмите кнопку Script Wizard и выберите *Script Output – Error*/*Info Dialog – Application Log – Add Message – Free Text*. Результат этого действия показан в листинге 6.5.



292



Listing 6.5 Free Text

Объект журнала приложения должен быть создан для добавления в него сообщений и вывода результата. В примере, приведенном в разделе 6.3, вы увидите, как создать свой прикладной журнал *log* в методе *INIT* и вывести его на экран в методе *END*.

Использование Z-классов в методе SCRIPT

Мастер сценариев содержит множество полезных классов, которые могут делать практически все, что только можно придумать, когда речь идет об управлении отладчиком. Однако вы обязательно столкнетесь с ситуацией, когда вам нужно будет сделать что-то, о чем не сказано. Когда этот день наступит, это не проблема: поскольку вы находитесь в обычном ABAP-методе, вы можете использовать свои собственные классы.

Допустим, вы хотите сохранить журнал приложения в базу данных. Стандартный CL\_TPDA\_SCRIPT\_MESSAGES не дает вам такой возможности, поэтому вместо того, чтобы пользоваться мастером, вы просто используйте методы класса, у которого есть такая возможность, возможно, реализующего интерфейс IF\_RECA\_MESSAGE\_LIST. Обычно нельзя создать подкласс стандартного класса SAP, например, CL\_TPDA\_SCRIPT\_MESSAGES, потому что 99,99% времени стандартные классы SAP устанавливаются на *final*. Поэтому создание собственных Z-классов с теми же интерфейсными методами в большинстве случаев является правильным решением.

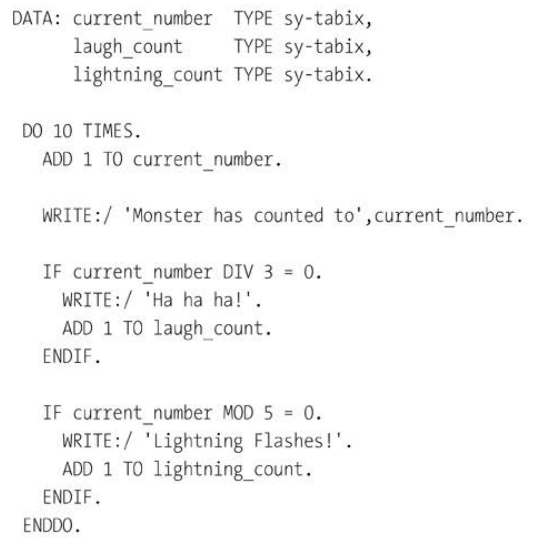
## 6.3 Кодирование методов INIT и END

Чтобы проиллюстрировать процесс написания сценария отладчика, давайте рассмотрим еще один пример - на этот раз тот, в котором задействованы методы INIT и END, а также метод SCRIPT. У вас есть программа, в которой монстр считает до десяти, и каждый раз, когда монстр доходит до числа, которое делится на три, он должен сказать: "Ха-ха-ха". Каждый раз, когда он доходит до числа, кратного пяти, сверкает молния.

293

Пользователи жалуются, что не хватает смеха и молний. Звучит фамильярно-знакомо, верно? Вы, вероятно, сталкиваетесь с почти такой же ситуацией в своей повседневной работе постоянно.

Пример программы показан в листинге 6.6. Вы сразу же заметите ошибку, которая заключается в том, что программа использует команду DIV вместо MOD: tut, tut, tut.

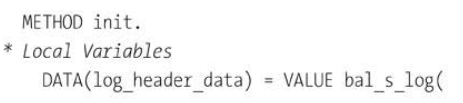


Listing 6.6 Laughing Program with Error

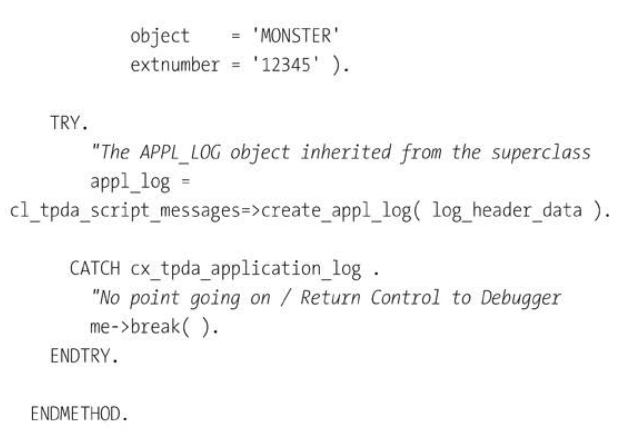
Хотя вы уже знаете, в чем заключается ошибка, представьте, что вы в затруднительном положении; вы просто не можете понять, в чем дело. Вместо того, чтобы перебирать программу по одной строке

за раз в поисках места ошибки, напишите сценарий отладчика, который покажет вам журнал приложения, анализирующий поток программы и сообщающий вам, что не так.

Установите специфическую для пользователя точку останова в самом начале программы, а затем выполните ее в вашей системе разработки. Оказавшись в отладчике, перейдите на вкладку Script и перейдите к методу *INIT*, где с помощью мастера сценариев вставьте шаблон для создания журнала приложения (листинг 6.7).



294



Listing 6.7 INIT Method

Мастер создания сценариев отладчика отстает от времени

Мастер генерирует код для вызова функциональных методов, используя довольно устаревший стиль по сравнению с тем, что доступно в языке ABAP в наши дни. В листинге 6.7 имеет обновленный синтаксис, отражающий современную практику.

Метод INIT запускается только один раз при выполнении сценария отладчика, поэтому это лучшее место для создания журнала приложения. Обратите внимание, что экземпляр журнала приложения называется APPL\_LOG и является переменной-членом суперкласса сценария отладчика, запускаемого вами класса сценария отладчика (проще говоря, вам не нужно создавать переменную для журнала приложения).

Затем нужно установить несколько точек наблюдения, чтобы сценарий отладчика выполнял метод SCRIPT каждый раз, когда изменяется одна из двух интересующих вас переменных (рис. 6.8 и рис. 6.8).

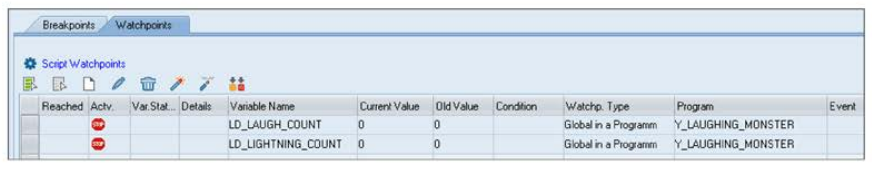


Figure 6.8 Laughing Example: Watch point Settings

295

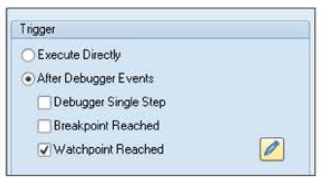
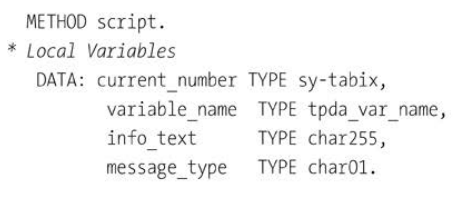
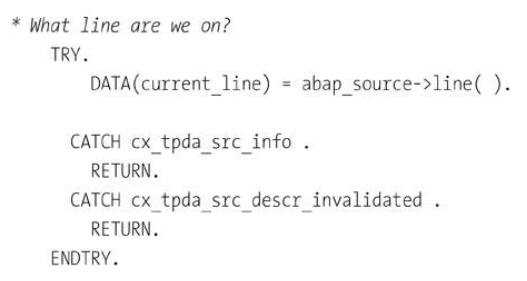
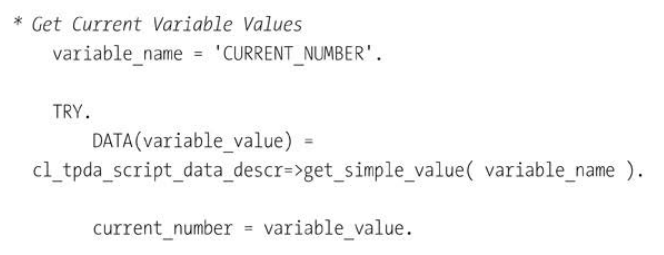


Figure 6.9 Laughing Example: Trigger Settings

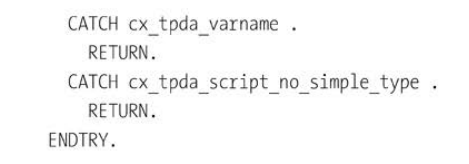
Далее, когда выполняется метод SCRIPT, необходимо выяснить, какая из сторожевых точек сработала, опросить текущие значения переменных и, наконец, записать некоторую информацию в журнал. На каждом этапе вы используете Мастер сценариев для импорта шаблона и результат выглядит как в листинге 6.8.

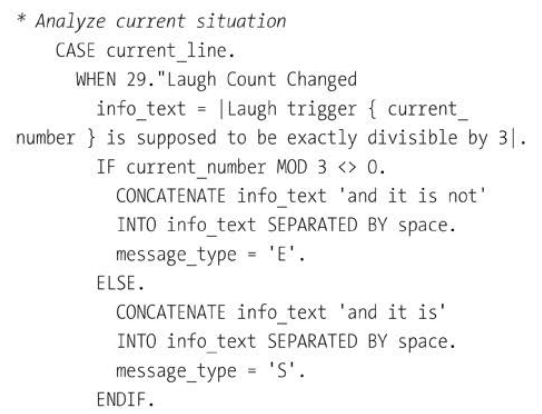


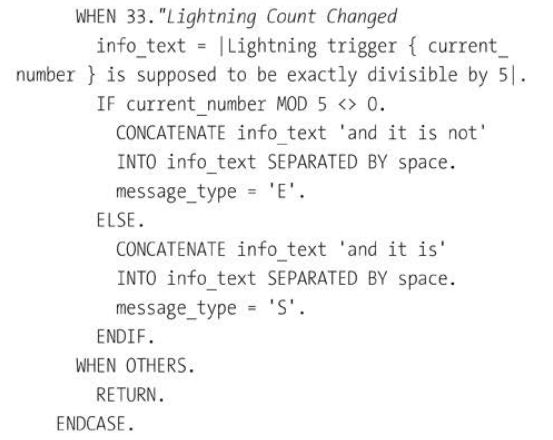


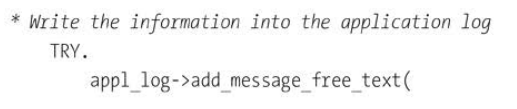


296

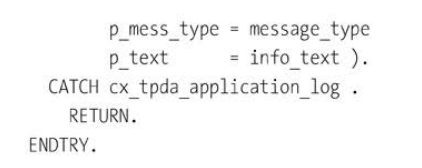


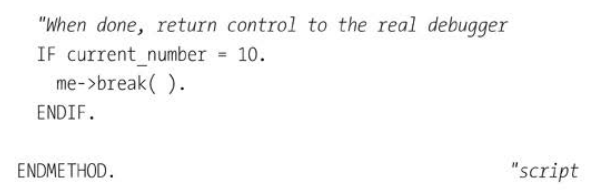






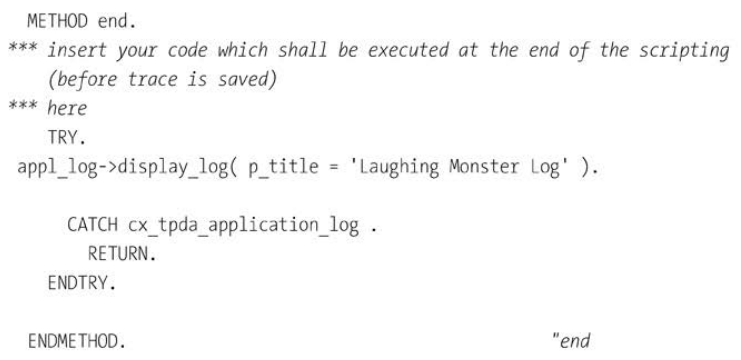
297





Listing 6.8 SCRIPT Method for Laughing Example

После завершения работы сценария вызывается метод END, и именно здесь вы отобразите журнал приложения. В листинге 6.8 вы заметите, что в конце метода SCRIPT есть некоторая условная логика, которая заключается в том, что после достижения последней ожидаемой точки наблюдения, вы принудительно устанавливаете точку останова и возвращаете управление в обычный отладчик. Это необходимо для того, чтобы перед лицом разработчика появилась кнопка End Script. Программист должен нажать эту кнопку; в противном случае метод END никогда не будет выполнен (Листинг 6.9). (Этот факт кажется очень странным; можно было бы ожидать, что метод END будет выполняться в конце сценария, как метод INIT автоматически запускается в начале сценария. Но в реальной жизни вам нужно нажать на кнопку, поэтому кнопка должна быть нажата).



Listing 6.9 End Script

298

Теперь выполните проверку синтаксиса вашего сценария, чтобы убедиться, что все в порядке и затем вы можете сохранить ваш сценарий в базе данных с помощью кнопки Save As (Сохранить как); когда вы нажмете кнопку появится окно, показанное на рисунке 6.10. Таким образом, если вам понадобится выполнить отлаживаемую программу пятьсот раз подряд, вы просто нажмете кнопку Load Script каждый раз, чтобы перейти к сценарию отладчика, а не писать сценарий заново. (Вы заметите, что сценарии отладчика можно также сохранять в локальных файлах и затем загрузить их в другую систему SAP).



Figure 6.10 Saving Debugger Script

И последнее, но не менее важное: запустите свой сценарий, нажав кнопку Start Script. Вы не сможете увидеть, что что-то происходит, пока не будет достигнута точка в вашем сценарии, в которой происходит прерывание и управление возвращается в обычный отладчик. Когда это произойдет, вы увидите выбор из двух кнопок для нажатия {Рисунок 6.11).

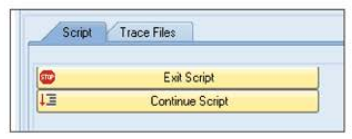


Figure 6.11 Exit or Continue Script

Если вы нажмете кнопку Continue Script, то ничего не произойдет; программа завершится нормально. Если вы нажмете кнопку Exit Script, то сценарий действительно завершится, но сначала выполняется метод END, в результате чего отображается журнал приложения, как показано на рисунке 6.12 (журнал отображается из-за некоторого кода в методе END (листинг 6.9), благодаря которому это происходит).

299

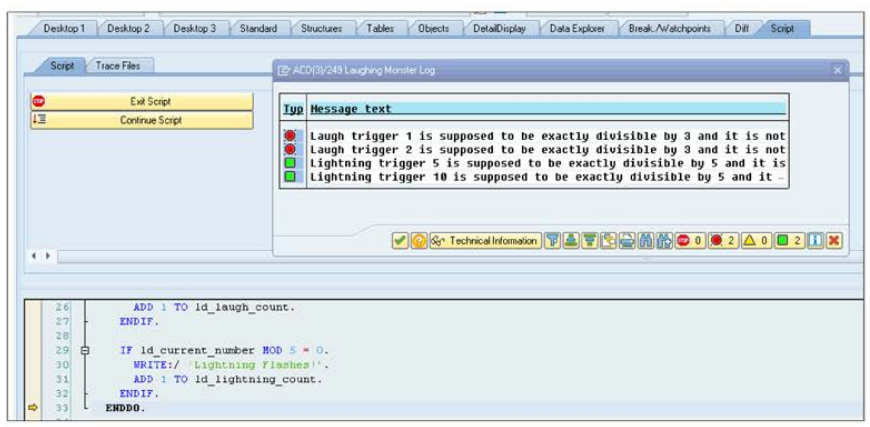


Figure 6.12 Application Log in Debugger Script

Одна из замечательных особенностей журнала приложений заключается в том, что вы можете вложить сообщения в древовидную структуру. Поэтому, если вы регистрируете колоссальное количество шагов, вы можете установить начальное отображение на очень высоком уровне, с красным пятном на верхнем уровне, если какой-либо из нижних шагов вызвал ошибку. В этом примере программист может посмотреть на строки ошибок в приложении *log* и сразу же увидеть, что первые две точки срабатывания - это числа один и 2 вместо 3, 6 и 9. Тогда причина ошибки становится очевидной, программист меняет DIV на MOD, и все в порядке.

Наконец, было бы упущением не упомянуть, что SAP ожидает от вас записи отладочной информации в файлы трассировки, а не в журнал приложения предоставляет вам различные опции записи трассировки для достижения этой цели.

## 6.4 Примеры реального мира

Если при взгляде на список опций, предоставляемых мастером сценариев, вы почувствуете парадокс выбора - вы можете сделать так много, что не знаете, с чего начать, и в итоге вообще не используете сценарии отладчика - не волнуйтесь. Ниже приведены некоторые советы как можно использовать сценарии отладчика в реальном мире. Мы начнем с некоторых общих примеров, а затем перейдем к более подробному рассмотрению того, как это может помочь при модульном тестировании:

300

- **Отслеживание каждой строки, выполняемой программой**

Этот пример вы чаще всего встречаете в стандартных примерах SAP о сценариях отладчика. Существует множество предопределенных сценариев отладчика, поставляемых компанией SAP, одним из которых является RSTPDA\_SCRIPT\_STATEMENT\_TRACE, который имеет только одну команду в методе SCRIPT. Это позволяет записать в файл трассировки каждую строку каждой отдельной программы, которая выполняется, в порядке выполнения этих строк.

Вы можете сделать копию таких предопределенных сценариев и изменить их как угодно, чтобы список результатов можно было фильтровать по точкам останова, точкам наблюдения или по любым пользовательским критериям, которые вы можете придумать.

- **Пропуск проверок полномочий**

Эксперт SAP Брайан О'Нил написал статью в блоге SAP Community, в которой он привел пример распространенной и утомительной проблемы при отладке. Иногда программист может не иметь полномочий на все в тестируемой программе.

Чтобы обойти эту проблему, традиционно нужно было останавливаться на каждом состоянии AUTHORITY-CHECK и, если проверка не удалась, вручную устанавливать SY-SUBRC в ноль, чтобы можно было продолжить.

Это быстро надоедает. С помощью сценария отладчика вы можете указать отладчику автоматически изменять значение SY- SUBRC на ноль сразу после каждой AUTHORITY-CHECK.

- **Создание точки наблюдения для символа поля**

Это еще один совет от Брайана О'Нила. В стандартном отладчике вы не можете иметь сторожевую точку, которая останавливает отладчик при изменении значения символа поля. С помощью сценариев отладчика он нашел способ сделать это возможным.

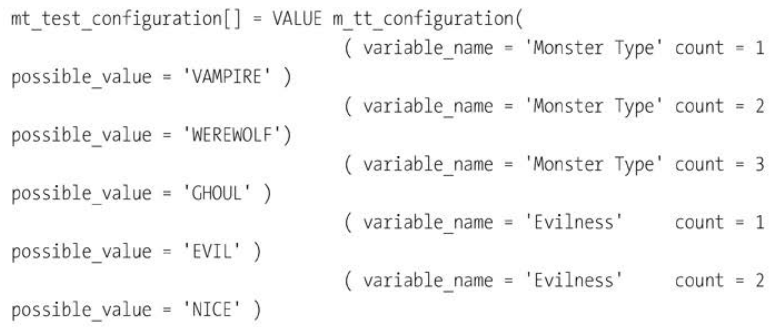
Вы увидите, что тема модульного тестирования поднимается практически в каждой главе этой книги из-за её центральной роли в новой модели программирования ABAP. Что касается отладчика в целом и отладочного скриптинга в частности, прекрасный пример Alexander Geppart можно найти в блоке "Рекомендуемое чтение" в конце этой главы. Там вы увидите список блога сообщества SAP, в котором рассказывается о том, как использовать отладчик для ускорения создания тестовых данных.

Чтобы представить некоторый контекст, бизнес-проблема в этом примере такова: часто вы расследуете ошибку и хотите создать модульный тест в разработке (естественно), который точно отражает ситуацию в производстве, чтобы вы могли быть на 100% уверены, что, когда ваш тест пройдет в разработке, измененный код устранит проблему в производстве.

Это требует, чтобы вы каким-то образом извлекли живые значения данных, которые вы можете видеть в отладчике - это могут быть многие строки внутренней таблицы или все поля очень длинной структуры (например, структуры, которую нужно заполнить для вызова BAPIs).

301

Затем при кодировании модульного теста вы вручную вставите все эти значения в test double, как показано в листинге 6.10.



Listing 6.10 Inserting Values into Test Double

В блоге Александра Геппарта и его коллеги написали программу для извлечения таких данных из отладчика и экспортировать их в текстовый файл в точном требуемом формате (оператор VALUE), чтобы вы могли просто вырезать и вставить текстовый файл в ваш тестовый код. На сайте проблема заключалась в том, что, хотя фреймворк отладчика очень хорошо спроектирован и расширяем, она расширяема только для разработчиков SAP, а не для нас, надоедливых клиентов. Поэтому было не так-то просто добавить свои собственные инструменты.

Поскольку пользовательских выходов не было, очевидным решением казалось использовать механизм усовершенствования, чтобы добавить код для нового инструмента извлечения в стандартный код SAP. Это очень легкая модификация стандартного SAP, но это тем не менее, модификация, и вы должны всегда стараться избегать ее, если только это не необходимо на 100%.

Ответ здесь заключается в том, что вы можете написать такой инструмент извлечения, используя отладчик сценариев; как вы видели, у вас есть доступ к полному набору команд ABAP, поэтому вы можете сделать то же самое, что и в улучшении. Это устраняет необходимость модифицировать стандартный SAP.

## 6.5 Резюме

Эта глава началась с признания того, что сидеть перед отладчиком - это как смерть и налоги - всегда с нами - и может быть довольно скучным и отнимать много времени. Вы

302

рассмотрели, как писать скрипты отладчика в ABAP для автоматизации задач ручной отладки, и затем рассмотрели несколько практических примеров того, как это можно использовать в реальной жизни.

Теперь, когда вы поняли концепцию, в следующий раз, когда вы окажетесь в отладчике - это может быть завтра; это может быть через десять минут после начала нового года - сделайте шаг назад, посмотрите, на то, что вы делаете, со стороны, и посмотрите, не делаете ли вы одни и те же шаги снова и снова. Если да, то вам, вероятно, поможет отладочный скриптинг. Прошли те времена, когда

вы проводили целый день в отладчике, гоняясь за одной проблемой.

Важно отметить, что сценарии отладчика не только могут автоматизировать ручные действия в отладчике, но также может заставить отладчик делать то, на что он раньше не был способен.

Это подводит вас к концу главы об инструментах программирования, которые можно было бы назвать помощниками. Далее вы перейдете к программированию бизнес-приложений, которые и есть эти инструменты. В новом мире SAP есть три столпа новой модели программирования. Первый из этих столпов включает программирование на уровне базы данных, в основном с использованием представлений CDS, и именно этому посвящена следующая глава!

Рекомендуемое чтение

• ABAP Debugger Scripting: Basics

https://biogs.sap.com/2070/72/74/abap-debugger-scripting-basics/

(Stephen Pfeiffer)

• Skip the Authority Check with the ABAP Debugger Script

https://biogs.sop.com/2073/03/22/skip-the-authority-check-with-the-abap-debugger-script/ (Brian O'Neill)

• How to Create a Watch point for a Field Symbol in the ABAP Debugger

https://biogs.sap.com/2073/03/08/how-to-create-a-watchpoint-for-a-field-symbol-in-the-abap-debugger /(Brian O'Neill)

• ABAP Debugger Enhancement or How to Speed Up Your Test Data Creation Process

https://biogs.sap.com/2077/72/17/abap-debugger-enhancement-or-how-to-speed-up-your-test-data-creation-process/ (Alexander Geppart)

303