Р10-2017-88

А. С. Кирилов

# ЭВОЛЮЦИЯ МОДУЛЯ ИНТЕРПРЕТАТОРА В ИНСТРУМЕНТАЛЬНОМ ПРОГРАММНОМ КОМПЛЕКСЕ **SONIX**+

Кирилов А. С. Р10-2017-88

Эволюция модуля интерпретатора
в инструментальном программном комплексе SONIX+

Программный комплекс SONIX+ разработан в качестве унифицированного программного обеспечения для управления экспериментом на нейтронных спектрометрах. В настоящее время он внедрен на спектрометрах реактора ИБР-2 в ЛНФ ОИЯИ, а также в ряде других центров РФ — всего около 20 инсталляций. В качестве языка управления экспериментом (скрипта) в комплексе выбран язык программирования Python. Модуль интерпретатора является его важнейшей частью. Этот выбор позволил не только описывать особенности процедуры конкретного спектрометра, но и включить элементы предварительной обработки результатов измерений в скрипт. Работа посвящена эволюции модуля интерпретатора скрипта — важнейшего элемента комплекса, отражающей рост сложности как самих спектрометров, так и методики проведения измерений на них.

Работа выполнена в Лаборатории нейтронной физики им. И. М. Франка ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2017

Kirilov A. S. P10-2017-88

Evolution of the Interpreter Module
in the Instrumental Software Complex SONIX+

The SONIX+ software complex is designed as a unified control software for neutron spectrometers. At present, it is installed at the instruments of the IBR-2 reactor at the FLNP, JINR, as well as in a number of other centers of the Russian Federation — about 20 installations alltogether. The Python programming language is selected as the control language for scripting experimental procedure in the complex. The interpreter module is its most important part. This allowed not only to describe the instrument procedure specifics but also to include elements of preliminary processing of the results of measurements in a script. The work is devoted to the evolution of the script interpreter module — the most important element of the complex, reflecting the increasing complexity of both the instruments themselves and the measurement methodology.

The investigation has been performed at the Frank Laboratory of Neutron Physics, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2017

## ВВЕДЕНИЕ

Принцип интерпретации программы эксперимента (скрипта) плодотворно используется во многих современных системах управления спектрометрами [1]. Подобная схема построения обеспечивает максимальную адаптацию системы управления к методике эксперимента и составу используемого оборудования. Дополнительным преимуществом «скриптовых» систем перед системами с полностью запрограммированным диалогом является возможность самостоятельного участия пользователей в корректировке программы управления.

В настоящее время при выборе скриптовых языков преимущество отдается универсальным языкам программирования (Tcl, Python и т. д.) [2–8], которые переживают период бурного развития. Опыт эксплуатации программного комплекса SONIX [9, 10] на спектрометрах реактора ИБР-2 также подтвердил этот тезис.

## ОСНОВНЫЕ ТРЕБОВАНИЯ ПРИ ВЫБОРЕ ИНТЕРПРЕТАТОРА (ИНТЕРПРЕТИРУЕМОГО ЯЗЫКА)

Под интерпретируемым языком, или *скриптом*, будем понимать сам язык, универсальный или специализированный, например, Tcl, Python и проч. Под *интерпретатором* (некоторые используют термин «sequencer») понимается программа, интерпретирующая скрипт. Под клиентом интерпретатора, или просто *клиентом*, понимается программа, реализующая пользовательский интерфейс при работе с интерпретатором. Следует отметить, что для языков собственной разработки функции интерпретатора и клиента могут быть реализованы в одной программе [7].

Скрипт в комплексе SONIX+ имеет три уровня иерархии:

— описание набора команд устройств;

— библиотеку операций спектрометра;

— библиотеку пользовательских скриптов.

На самом нижнем уровне все устройства системы описаны как классы на языке Python с присущими атрибутами и набором команд. В отдельном файле <имя_спектрометра>_python_configuration.py перечислены все устройства, входящие в состав конкретного спектрометра. Эти файлы составляются программистом.

Библиотека операций спектрометра включает процедуры, характерные для конкретной методики измерений на спектрометре. Их будем называть командами спектрометра. Библиотека составляется программистом после консультаций с физиками, которые впоследствии могут редактировать или дополнять библиотеку самостоятельно. Команды спектрометра должны быть удобны и понятны для пользователя спектрометра.

Библиотеку пользовательских скриптов образуют задания на конкретные эксперименты. Они могут быть созданы пользователем вручную с помощью текстового редактора или с помощью программ генераторов, которые по заданию пользователей могут быть написаны программистами.

Интерпретатор (в сочетании с клиентом) должен обеспечивать:

— запуск и остановку измерения;

— временную приостановку измерения с возможностью ручного ввода команд и продолжения с прерванного места;

— информацию о текущей интерпретируемой строке.

Программа-клиент должна быть интегрирована в систему управления и прежде всего взаимодействовать с остальными модулями измерительного комплекса по принятым в нем правилам напрямую и/или через скрипт. При этом она должна быть независимой от специфики измерений, иметь, по возможности, простой, понятный интерфейс.

Язык Python [3] предоставляет разнообразные средства для встраивания интерпретатора во внешнюю программу на языке C. Прежде всего, это Pyhon/C API, который предлагает программисту широкий спектр возможностей, начиная от операций высокого уровня (интерпретации целого файла с помощью процедуры PyRun_SimpleFile()) и кончая определением новых типов данных. Однако PyRun_SimpleFile() не обеспечивает временной приостановки. Построение собственного построчного отладчика, упомянутого в Extending/Embedding FAQ, тоже не решает проблему, если исполняемый скрипт содержит циклы, условные операторы и т. д.

## ПЕРВЫЙ ВАРИАНТ ИНТЕРПРЕТАТОРА

Начальный вариант интерпретатора для комплекса SONIX+ был реализован по технологии отладчика. Для этой цели на основе стандартного класса заготовки отладчика в Python pdb был составлен класс pideb. Главные отличия нового отладчика от стандартного состоят в добавлении механизмов:

— выдачи информации о текущей позиции (файл – процедура – номер строки);

— выдачи списка значений отслеживаемых переменных;

— передачи ручных команд для интерпретации;

— переопределения ввода/вывода;

— управления визуализацией текущих строк с помощью переменной __TRACE__;

— введения флага перехода в пошаговый режим;

— введения флага принудительной остановки.

Пользовательский интерфейс Pi.exe реализован как многодокументное приложение на Visual C++. Для связи интерфейса и pideb был разработан протокол взаимодействия, реализованный в виде библиотеки dbg_comm.dll.

Для управления визуализацией текущих строк (соответственно, открытием новых окон с текстами соответствующих файлов) был реализован специальный механизм с помощью переменной __TRACE__. Для процедур, в которых определена эта переменная, выдача информации о текущей позиции, сброс буферов вывода и значений отслеживаемых переменных производится после выполнения каждой строки. Если значение переменной больше 0, то ее выполнение возможно в пошаговом режиме и, соответственно, переход в этот режим осуществляется по команде интерфейса Stop. Определение переменной __TRACE__ должно быть произведено обязательно в первой исполняемой строке процедуры. Проверка синтаксиса реализована через компиляцию файла и проверку правильности отступов с помощью TabNanny.

## ИНТЕРФЕЙС С ПОЛЬЗОВАТЕЛЕМ

Программа Pi имела меню, две панели инструментов, поле для ручного ввода команд, индикатор состояния, три окна для представления текущего состояния измерения и поле статуса. Примерный вид окна Pi показан на рис. 1.

Текущее состояние измерения представлено:

— индикатором состояния;

— окном визуализации программного кода;

— окном Watch для просмотра избранных переменных;

— окном Output;

— строкой состояния.

Поле ручного ввода команды активизируется сразу перед запуском скрипта и во время остановок (приостановок). Можно вводить любые строки на языке Python, которые будут выполнены в текущем контексте.

Программа использовалась на спектрометрах НЕРА-ПР и РЕМУР и до остановки реактора ИБР-2 на модернизацию в 2007 г.

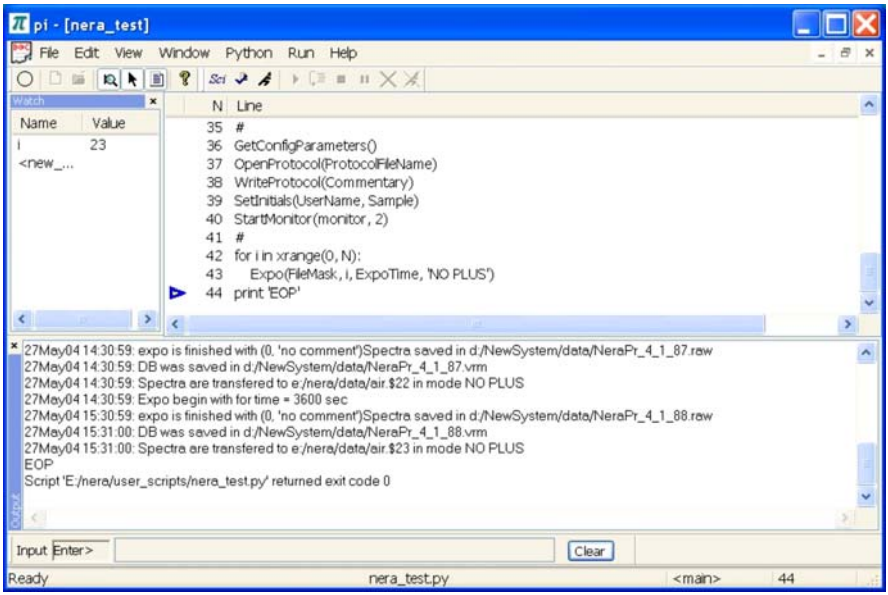


Рис. 1. Примерный вид окна интерпретатора

## ВТОРОЙ ПОДХОД

В процессе эксплуатации первой версии проявились недостатки реализации интерпретатора на основе отладчика. Перенос частичной обработки спектров в скрипт значительно усложнило последний. Ведение библиотеки типовых операций пользователя не позволяет полностью скрыть от пользователя эту сложность, даже при использовании флага __TRACE__. Отладчик — инструмент скорее программиста, а не физика. Пользователю «не интересно» следить за построчным выполнением кода, тем более если он сложен. Для многих достаточно подготовить задание и запустить его до ошибки или конца измерения. Кроме того, выполнение скрипта в режиме отладчика происходит на порядок медленнее. Это незаметно при обмене командами, но применение математической обработки многомерных массивов данных в скрипте существенно увеличивает время его выполнения. Эти обстоятельства предопределили необходимость переработки начальной версии.

Для обеспечения максимальной скорости интерпретации было предложено использовать метод контрольных точек. Для этого в команды спектрометра вставляется вызов особой процедуры contCPoint(n_level, comment), в которой может быть выполнено вмешательство в процесс выполнения скрипта, т. е. остановка, введение команд вручную, продолжение эксперимента.

Для удобства отладки введена иерархия контрольных точек по номеру уровня. Например, уровень типовых операций пользователя имеет уровень 1, а элементарные операции для устройств — уровень 100. Изменяя разрешенный уровень реакции, можно изменять глубину «погружения» в скрипт. Контрольные точки с уровнем 0 вызывают прерывание всегда. Они предназначены для обработки фатальных ошибок.

## СОСТАВ И СХЕМА ВЗАИМОДЕЙСТВИЯ

На рис. 2 приведена схема взаимодействия отдельных компонентов новой версии интерпретатора. Программа Is.exe реализована на языке C++ в виде стандартного сервера SONIX+, который доступен как обычное устройство со своим статусом, параметрами и набором команд. Через коммуникационную библиотеку dbg_comm2.dll, составленную как расширение языка Python, он взаимодействует с системной частью скрипта, оформленной в файле cont_points.py. В свою очередь, программа эксперимента (скрипт на языке Python) импортирует cont_points.py через библиотеку команд спектрометра. Для обмена сообщениями введены события Windows «EnterCommand» и «OutputIsReady». Синхронизация операций чтения/записи реализована с помощью механизма критических секций.

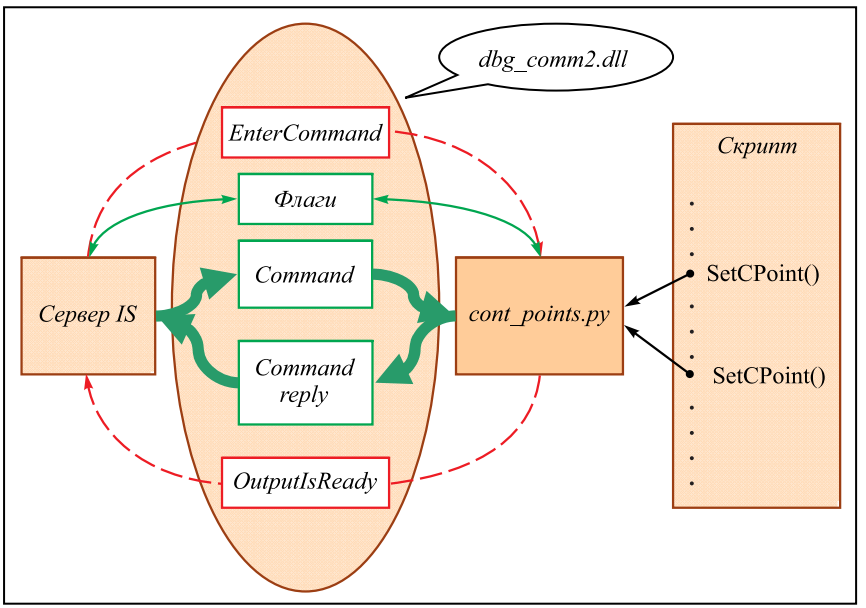


Рис. 2. Состав и схема взаимодействия компонентов SONIX+ интерпретатора второй версии

Для пользователя разработан стандартный MFC GUI is_client.exe. Кроме того, набор клиентских операций на C++ был оформлен классом CIsWrapper для возможного применения в других программах. Позднее стандартный GUI был также реализован с помощью PyQt [11].

Эта версия успешно использовалась с момента пуска реактора ИБР-2 в 2011 г. вплоть до настоящего времени.

## ТРЕТЬЯ ВЕРСИЯ ИНТЕРПРЕТАТОРА

Основной причиной новой модернизации интерпретатора стало активное применение в скрипте пакета Numerical Python (numpy) [10], эффективного средства для математической обработки многомерных данных. Выяснилось, что при использовании функций Py_Initialize()/Py_Finalze() из Python API для перезапуска Python рабочие буферы в динамической памяти компьютера очищаются не полностью. Это вызывает ошибку при повторном запуске скрипта в момент импортирования numpy. В связи с тем, что операции по обработке данных в скриптах постоянно усложняются, решение проблемы с numpy представлялось очень актуальным.

Рассматривались два варианта решения этой проблемы. В первом случае предполагался отказ от использования Py_Initialize()/Py_Finalze() с заменой на прямой вызов Python.exe с исполняемым скриптом в качестве параметра. Однако в этом случае возникли серьезные трудности с возвратом реакции на сигналы Suspend и Break, используемые в SONIX+ для приостановки или прекращения измерения. Причиной этого является то, что Python.exe не поддерживает механизм межмодульного общения в SONIX+.

Другим вариантом решения стала перезагрузка модуля интерпретатора при каждом запуске скрипта на исполнение. В этом случае очистка динамической памяти операционной системой выполняется полностью и ошибка с повторным импортированием numpy не проявляется. Данный вариант и был реализован в третьей версии интерпретатора.

После выполнения скрипта или его аборта модуль интерпретатора выгружается автоматически, а в момент запуска нового скрипта — загружается также автоматически. Важно отметить, что процесс загрузки происходит очень быстро и незаметно для пользователя.

Переработка интерпретатора позволила выполнить и ряд других усовершенствований за счет упрощения структуры и ликвидации функциональности, не востребованной пользователями. В частности, была удалена коммуникационная dbg_comm2.dll. Все актуальные значения статуса и параметров интерпретатора сосредоточены только в дескрипторе модуля в базе данных (БД) Varman (табл. 1).

**Таблица 1. Дескриптор интерпретатора**

| Имя | Назначение |
|---|---|
| Параметр | |
| abort_flag | 1 — установить режим экстренного окончания измерения |
| manual_command | Буфер ручной команды |
| non_stop_flag | 1 — непрерывное выполнение скрипта, 0 — пошаговое |
| rlevel | Минимальный уровень реакции контрольных точек |
| stop_exit_flag | 1 — установить режим Stop&Exit |
| user_script | Путь к выполняемому файлу скрипта |
| Статус | |
| alevel | Уровень последней контрольной точки |
| command | Текущая команда |
| diagn_line | Строка диагностики |
| error_flag | Флаг ошибки |
| file_name | Путь к файлу скрипта (может отличаться от user-script, если контрольная точка находится в другом файле) |
| line_num | Номер текущей строки |
| proc_name | Имя исполняемой процедуры |
| value | Состояние интерпретатора |

Изменения параметров в статусе дескриптора выполняются через механизм коротких команд, а параметры дескриптора выполняются из файла cont_points.py. Поскольку статус и параметры дескриптора хранятся в разных переменных БД, синхронизации обращений не требуется.

Изменены набор состояний (табл. 2) и состав команд (табл. 3). Преемственность с прежним набором не сохранена. Текущее состояние (None, Ready, Busy, Susp или Wait) прямо указывается в статусе. Флаги abort_flag, error_flag, non_stop_flag и stop_end_exit_flag вынесены в отдельные переменные статуса и доступны непосредственно. В предыдущей версии часть из них объявлялась глобальными переменными Python, что затрудняло отладку скрипта.

Изменен набор команд. Файл remote_control2.py для управления интерпретатором из пользовательского интерфейса исправлен.

Флаг __CheckParametersFlag__ для управления процессом проверки скрипта не внесен в статус и остается глобальной переменной Python. В файл common_lib.py добавлены функции так, чтобы изменения этого флага всегда отражались в протоколе SONIX+. Оказалось, что, помимо ошибки с numpy,

**Таблица 2. Набор состояний интерпретатора**

| Состояние | Описание |
|---|---|
| None | Модуль интерпретатора выгружен |
| Ready | Интерпретатор загружен и готов к работе (сразу после перезагрузки) |
| Busy | Процесс интерпретации выполняется |
| Susp | Процесс интерпретации приостановлен до отмены suspend |
| Wait | Процесс интерпретации остановлен. Ожидается команда от пользователя |

**Таблица 3. Набор команд интерпретатора**

| Команда | Описание |
|---|---|
| pystart (file_name) | Запуск эксперимента со скриптом file_name. Переход в состояние Busy |
| pystop | Остановиться после выполнения текущей команды |
| pystep | Выполнить следующую команду |
| pyabort | Немедленно прервать эксперимент и выгрузить интерпретатор. Переход в состояние None |
| pyisexit | Выгрузить модуль интерпретатора. Переход в состояние None |
| pycont | Продолжить эксперимент после состояния Wait |
| pysuspend | Немедленно приостановить текущую операцию. Переход в состояние Susp |
| pycontinue | Продолжить выполнение из состояния Susp |
| pycheck_code (file_name) | Выполнить проверку скрипта file_name |
| pyset_frame_parameters (fname, name, lino, line, nlevel, comment) | Записать в дескриптор параметры фрейма |
| pystop_and_exit | После завершения текущей команды закончить эксперимент и выгрузить модуль интерпретатора (функция Stop&Exit) |
| pysset_state (state) | Записать состояние в дескриптор |
| is_load | Загрузить/перегрузить модуль интерпретатора |

проверка корректности скрипта могла давать неверный результат. Перезапуск интерпретатора скрипта решил и эту проблему.

При разработке третьей версии ряд параметров, существенных для комплекса в целом, введен в реестр компьютера. Специфика спектрометра в значительной степени сосредоточена в следующих файлах: библиотеке операций

спектрометра (<instrument_name>_lib.py), конфигурации устройств для загрузки (<instrument_name>_instrument_configuration.py) и конфигурации устройств для скрипта (<instrument_name>_python_configuration.py). Для того чтобы сделать пользовательский интерфейс (GUI) полностью независимым от этой специфики, имена этих файлов заносятся в реестр при загрузке комплекса. Для удобства все операции с реестром собраны в файле winreg_operations.py.

## РЕАЛИЗАЦИЯ ФУНКЦИИ Stop&Exit

По просьбе пользователей рефлектометров в интерпретатор была добавлена функция Stop&Exit. Типичное измерение на этих спектрометрах организовано как двойной цикл с перебором состояний флипперов, анализом промежуточных результатов и их суммированием. Команда задает окончание измерения, отложенное до окончания внутреннего цикла и завершения окончательного суммирования. Связь между скриптом и GUI выполняется через переменную БД isExecutionFlag. В обычном состоянии ей присвоено значение «Goon», при выборе режима Stop&Exit значение меняется на «StopExit». Конкретная реакция на команду программируется в библиотеке операций спектрометра.

## ИЗМЕНЕНИЯ В GUI

Для работы с третьей версией интерпретатора были внесены коррективы в некоторые элементы (виджеты) GUI.

**Изменения в программе загрузки SONIX+.** Программа S+load_panel.pyw (рис. 3) предназначена для загрузки/выгрузки комплекса и проведения некоторых проверок. После выполнения загрузки комплекса и записи дополнительных параметров в реестр программа по просьбе пользователей автоматически проводит проверку наличия всех модулей в памяти компьютера и определяет,


Рис. 3. Интерфейс программы загрузки SONIX+


Рис. 4. Пример сообщения о результатах загрузки комплекса

для каких устройств в процессе загрузки по каким-либо причинам не установлена связь с соответствующими контроллерами. Пример результирующего сообщения приведен на рис. 4.

**Изменения в виджете Si.** Виджет Si предназначен для управления интерпретатором и контроля его работы. Он может быть использован и самостоятельно, и в качестве части унифицированного GUI. В виджет внесены несколько существенных изменений. Если в БД определена переменная isExecutionFlag, рисуется дополнительная кнопка StopExit (рис. 5), в противном случае она не рисуется (см. разд. «Реализация функции Stop&Exit»).

При выборе имени файла, как и было ранее, сначала выполняется проверка скрипта. В новой версии для этой проверки производится отдельный запуск Python с файлом cc_test.py, чтобы исключить возможную ошибку при повторных проверках скрипта после исправления. Результат проверки помещается в раздел реестра SCRIPT_CHECK. Для спектрометра может быть преду-

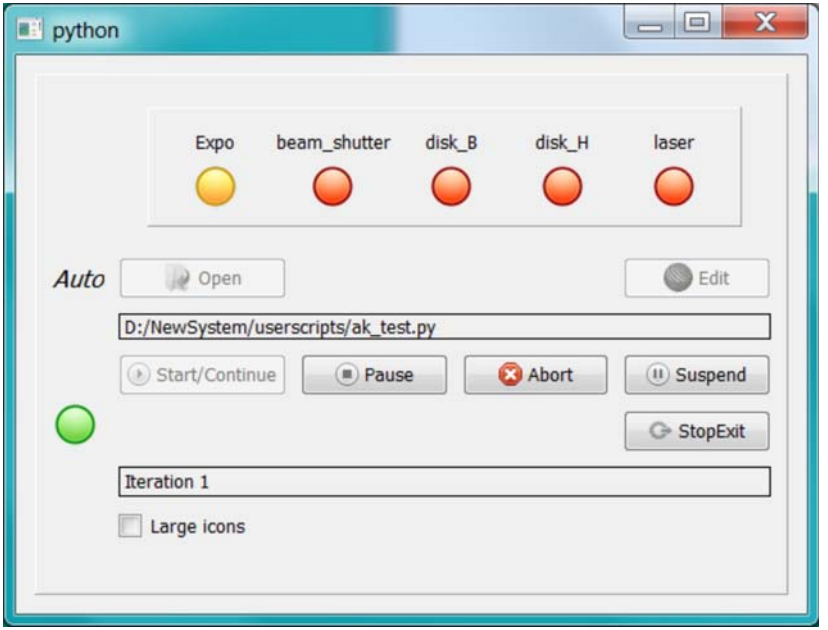


Рис. 5. Общий вид виджета Si с кнопкой StopExit

смотрена дополнительная проверка после основной. В состав библиотеки спектрометра для этого необходимо включить функцию scriptFileExtraTesting(filename). В результате проверок выдается общее сообщение. Например, для спектрометра ЮМО предусмотрены подсчет общего времени экспозиции и проверка корректности задания позиций детекторов. Результат этой проверки представлен на рис. 6.

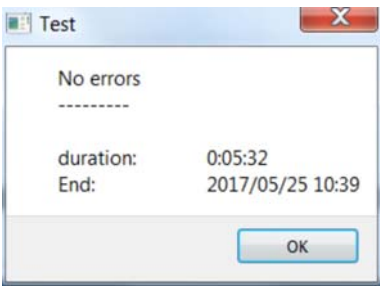


Рис. 6. Пример результата проверки скрипта для спектрометра ЮМО

## ЗАКЛЮЧЕНИЕ

Программный комплекс SONIX+ (SONIX) разработан в качестве унифицированного программного обеспечения для управления экспериментом на нейтронных спектрометрах, которое в настоящее время внедрено на спектрометрах реактора ИБР-2 в ЛНФ ОИЯИ, а также в ряде других центров РФ —

всего около 20 инсталляций. Модуль интерпретатора является его важнейшей частью. Длительная практика показала удачность выбора языка Python для программирования процедуры измерения. Последняя версия интерпретатора успешно прошла проверку на спектрометре ЮМО, начиная с осени 2017 г., и будет перенесена на все остальные спектрометры реактора ИБР-2 в ближайшее время.

## ЛИТЕРАТУРА


1. http://www.nobugsconference.org/
2. http://www.tcl.tk/
3. http://www.python.org
4. *Koennecke M.* http://lns00.psi.ch/sics/design/sics.html.
5. http://www.esrf.eu/computing/bliss/doc/bliss/
6. https://forge.frm2.tum.de/nicos/doc/nicos-master/
7. *Кирилов А. С., Хайниц Й.* Сообщение ОИЯИ Р13-97-161. Дубна, 1997.
8. *Kirilov A. S.* Script Interpreter in the Software Complex SONIX+. http://lns00.psi.ch/nobugs2004/.
9. *Кирилов А. С.* Новые версии программ юстировки и визуализации спектров для рефлектометров реактора ИБР-2 // Письма в ЭЧАЯ. 2016. Т. 13, № 1(199). С. 208–219.
10. http://numpy.org
11. *Кирилов А. С., Петухова Т. Б.* Набор компонентов для построения GUI в системах управления нейтронными спектрометрами на основе PyQ. Сообщение ОИЯИ Р10-2015-38. Дубна, 2015.


Получено 11 декабря 2017 г.

Редактор *Е. В. Григорьева*

Подписано в печать 28.12.2017.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 0,94. Уч.-изд. л. 1,16. Тираж 200 экз. Заказ № 59303.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@jinr.ru
www.jinr.ru/publish/