CA Unified Infrastructure Management

Developer White Paper

Transitioning from Distsrv to Automated Deployment Engine 2.0

ca technologies

Document Change History

Date	Version	Change
September 2014	Version 2.0	Initial draft of document for ADE 2.0, released with UIM Server 8.0.
November 2014	Version 2.0	Notice added.

NOTICE

ADE CALLBACKS ARE SUBJECT TO CHANGE.

The callbacks are very low level APIs that are primarily used internally by UIM Server. Future changes to callbacks may consist of implementation changes, including changes to the signature or the name of the callback. This will require that you modify scripts or other custom tools based on the callbacks.

CA will strive to minimize such impacts but reserves the right to do so, without advanced notice or formal deprecation periods. Changes to callbacks will be reflected in the documentation for each release.

Table of Contents

Overview

About this Document

This white paper intended to help UIM partners and administrators who have used distsrv callbacks in scripts or applications begin transitioning to ADE.

NOTICE

ADE CALLBACKS ARE SUBJECT TO CHANGE.

The callbacks are very low level APIs that are primarily used internally by UIM Server. Future changes to callbacks may consist of implementation changes, including changes to the signature or the name of the callback. This will require that you modify scripts or other custom tools based on the callbacks.

CA will strive to minimize such impacts but reserves the right to do so, without advanced notice or formal deprecation periods. Changes to callbacks will be reflected in the documentation for each release.

Distsrv and ADE

Distributing the UIM software throughout your infrastructure is an important aspect of the Unified Infrastructure Management solution. Prior to the release of UIM Server 8.0:

- The **distrv** probe maintained distributable packages, licenses, and alarm console maps, and transferred probe packages to or from robots.
- The **automated_deployment_engine** (ADE) probe enabled you to deploy robots in bulk.

With the release of UIM Server 8.0, the tasks performed by distsrv begin transitioning to the automated_deployment_engine (ADE) probe.

ADE v2.0

ADE v2.0 is a complete redesign of the distsrv probe in java code, designed with extensibility and scalability in mind. The main benefit of this release is that Admin Console uses ADE instead of distsrv. ADE 2.0 also provides improvements in:

- **Performance**
  - ADE is much faster than distsrv (on the order of 3 to 5 times faster for most operations).
  - The number of concurrent tasks that ADE can perform will scale according to the CPU resource provided to it. ADE is designed to effectively take advantage of CPU and memory resources in order to perform tasks quickly and efficiently.
- **Extensibility**
  - Because ADE was written from the ground up to replace the functionality and services provide by distsrv, the design can be more easily extended and enhanced for future business needs.

Functional Comparison

Functionality	ADE	Distsrv
Parallel Deployment	The number of probes that can be deployed concurrently is based on the number of threads ADE has available. ADE automatically deploys probes single-threaded to a single target and multi-threaded to multiple targets.	All packages are deployed serially regardless of the number of targets specified.
Archive Management	ADE manages an in-memory cache of the archive at any given time. Any time probes are added and removed from the archive by using callbacks, the cache is updated. This allows for quick lookups and quick retrievals.	During startup, distsrv unzips all available zip packages in the archive and stores the information about them. For any further lookups it has to unzip the package and update the stored information.
License Validation	ADE does not have the licensing functionality built it. In a future release, licensing will be moved to a component outside of ADE. This may be the hub or may remain in distsrv.	Licenses are managed and validated by distsrv and the hub.
Resource Management	ADE does on-demand zip unpacking, keeps the unpacked the files available on the file system. These resources are available to any and all future distributions. They are not limited to one job or one task, as many readers can access a file at a time. It does not unzip any package that it does not need.	Distsrv does on-demand unzip unpacking, but does not keep the resource for later. Partly due to its single threaded nature, distsrv uses a resource and then deletes it after distribution. This does not allow for any sharing between jobs and forces an unpack for each and every job.
Size	~8.0 MB	~3.0 MB
Package Forwarding	*As of most recent ADE 2.0 build, this is not provided. Plans are slated for UIM Server 8.0.*	Distsrv does package forwarding according to a set of rules. It tends to take up a lot of bandwidth and has been known to be under performant.

Callback Transition

The following table outlines the callback equivalents between ADE and distsrv. For further understanding and greater detail of the ADE callbacks, see the “Automated Deployment Engine Callbacks” section.

Functionality	ADE	Distsrv
Create archive package backup	archive_backup	archive_backup
Delete archive package	archive_delete	archive_delete
Download archive package	archive_get_start archive_get_next archive_get_end	archive_get_start archive_get_next archive_get_end
List contents of archive	archive_list	archive_list
Add package to archive	archive_put_start archive_put_next archive_put_end	archive_put_start archive_put_next archive_put_end
Deploy single archive package	deploy_probe	job_add
Retrieve job status	get_status	job_list or job_status
Cancel a job	cancel_job	job_cancel
Multi-package deployment	submit_job	job_add(multiple calls)

Automated Deployment Engine Callbacks

Archive Management

archive_backup(name,version)

Creates a backup of the specified archive with name and version.

Input Parameter	Type	Required	Description
name	String	Yes	Name of the package as present in “archive_list” callback.
version	String	Yes	Version of the package as present in “archive_list” callback.

Error	Cause
NimException.E_ERROR	Error occurred while attempting to create backup.
NimException.E_INVAL	Specified name or version does not match an entry in the archive.

archive_delete(name,version)

Deletes the archive with the specified name and version from the archive cache and file system.

Input Parameter	Type	Required	Description
name	String	Yes	Name of the package as present in “archive_list” callback.
version	String	Yes	Version of the package as present in “archive_list” callback.

Error	Cause
NimException.E_INVAL	Invalid name or version specified. The *name* and *version* parameters cannot be empty or null.
NimException.E_ERROR	Error occurred while attempting to delete specified archive package.

archive_list(name,version)

Lists the contents of the ADE archive cache. This is equivalent to distsrv archive_list with the distsrv detail field set to the highest setting (integer 3).

Input Parameter	Type	Required	Description
name	String	No	Name of the package to provide information for.
version	String	No	Version of the package to provide information for.

The return PDS for this callback contains an array of PDSs under the key "entry" with each representing an entry in the archive. For the sake of simplicity, the table below covers the contents of each archive entry PDS.

Return Value	Type	Description
name	String	Name as it appears in infrastructure manager or admin console.
description	String	Formal name of the package or a functional description.
group	String	Functional group the package belongs to.
author	String	Creator or maintainer of this package.
copyright	String	Date of copyright.
license_required	String	Indication of whether or not a license is required to deploy the package.
version	String	Version of the package as determined by the author.
date	String	Date the package was created.
build	String	Build number of the package.
file_name	String	Absolute path to the archive zip on the filesystem.

archive_get_start(name,version,buffer_size)

Starts the transaction for an archive download.

Input Parameter	Type	Required	Description
name	String	Yes	Name of the package as present in “archive_list” callback.
version	String	No	Version of the package as present in “archive_list” callback. No version specified is equivalent to highest version.
buffer_size	Integer	No	Size of the buffer to store downloaded bytes. This is used to receive the file in chunks.

Return Value	Type	Description
file_content	Byte	First chunk of data from the archive package.
id	String	ID is used for the initial call to archive_get_next. Subsequent calls to archive_get_next return an updated id.
read	Integer	Represents the amount of data read into the file_content buffer.

Error	Cause
NimException.E_ERROR	Invalid checksum generated for package. Unable to generate checksum for package specified. Unable to read from specified package.
NimException.E_NOENT	File specified does not exist.
NimException.E_INVAL	Arguments provided are invalid.

archive_get_next(id)

Continues the transaction for an archive download.

Input Parameter	Type	Required	Description
id	String	Yes	ID is used by ADE to continue the archive download transaction.

Return Value	Type	Description
file_content	Byte	Next chunk of data in the archive package being read.
id	String	Updated ID for the archive download transaction.
read	Integer	Amount of data read into the file_content buffer.

Error	Cause
NimException.E_ERROR	Unable to read data from the file specified by the id.
NimException.E_INVAL	Id provided was invalid.

archive_get_end(id)

Ends the transaction for an archive download.

Input Parameter	Type	Required	Description
id	String	Yes	The is used by ADE to identify and end the transaction.

Return Value	Type	Description
id	String	Final ID associated with the transaction.
read	Integer	The total amount of bytes read from the file.

archive_put_start(name,version,file)

Starts the transaction for an archive addition.

Input Parameter	Type	Required	Description
name	String	Yes	Name of the package to be uploaded.
version	String	Yes	Version of the package to be uploaded. In the past, distsrv used this field, but currently ADE only uses it when naming the temporary file.
file	String	No	*Currently this field does nothing. ADE provides the filename transparently.*
file_content	Byte	No	The initial chunk of data to be uploaded. This field does not need to be specified in the initial call.

Return Value	Type	Description
id	String	The ID associated with the upload transaction. Required to call “archive_put_next”.
written	Integer	Amount of data written out to the temporary file during this callback. This is only specified if “file_content” was specified in the initial call.

Error	Cause
NimException.E_ERROR	Unable to write data to package file. Could not find specified package file.
NimException.E_INVAL	Arguments provided are invalid. Arguments “name” and “version” cannot be empty or null. Version provided does not match the standard format. Specified file argument is not relative to the archive directory.

archive_put_next(id)

Continues the transaction for an archive addition.

Input Parameter	Type	Required	Description
id	String	Yes	Identifies the upload transaction.

Return Value	Type	Description
id	String	ID associated with the next phase of the upload. A new ID is returned each time this callback is called.
written	Integer	Number of bytes written during the call to “archive_put_next”.

Error	Cause
NimException.E_INVAL	Specified “id” is not valid.
NimException.E_ERROR	File specified by “id” does not exist. Unable to write to file specified by “id”. Argument “file_content” contains no data.

archive_put_end(id)

Ends the transaction for an archive addition.

Input Parameter	Type	Required	Description
id	String	Yes	Identifies the upload transaction.

Return Value	Type	Description
file	String	Absolute path to the new archive directory on the filesystem.

Deploying Packages

deploy_probe

Provides functionality similar to the “job_add” callback in distsrv. This will deploy a single package to a single robot.

Input Parameter	Type	Required	Description
package	String	Yes	Name of the package to be deployed.
version	String	No	Version of the package to be deployed. If no version is specified ADE selects the highest one.
robot	String	Yes	The target for the deployment. This should be the nimbus address for a robot.
update	String	No	Either “0” for do not update the package if exists or “1” for update the package regardless of if it exists. Default is “1”.
startAt	String	No	*Currently provides not additional functionality.*
jobname	String	No	Name of the job. Currently supports up to 255 characters.
job_description	String	No	Description of the job. Currently supports up to 255 characters.

Return Value	Type	Description
JobID	String	UUID identifier of the job created by ADE.

submit_job

Provides a way to specify a multi-package multi-robot job. This callback is also used to deploy robots, but is not covered in this whitepaper. To deploy more than one package to a single robot or to deploy multiple packages to multiple robots a table of PDSs with the key “probes” must be constructed. The parameters specified below the key “probes” are for each PDS within the table.

Input Parameter	Type	Required	Description
probes	PDS (table)	Yes	Table containing an entry for each PDS specified for the job.

An entry in “probes” table. All entries are retrieved with getTablePDSs(“probes”).

Input Parameter	Type	Required	Description
package	String	Yes	Name of the package to deploy.
version	String	No	Version of the package to deploy. If no version is provided, ADE selects the highest.
robot	String	Yes	Target robot for the deployment.
update	String	No	Either “0” for do not update the package if exists or “1” for update the package regardless of if it exists. Default is “1”.
startAt	String	No	*Currently provides no additional functionality.*

Note: The *jobname* and *job_description* keys can be provided at the same level as the *probez* key.

Return Value	Type	Description
JobID	String	UUID identifier of the job created by ADE. This is used by the “get_status” callback to retrieve the status of a job.

Retrieving Status

get_job_ids

Returns an array of Strings with each one representing aunique JobID that is currently being tracked by ADE.

Return Value	Type	Description
JobIDs	PDS(table)	Array containing all the JobIDs being tracked by ADE.

get_status

Retrieves the compiled status for a job created by ADE.

Input Parameter	Type	Required	Description
JobID	String	Yes	UUID identifier of the job created by ADE.

The PDS returned by "get_status" contains a multitude of information about the jobs and each task contained within. The PDS specified by the key "StatusTable" contains an entry PDS for each task within the job. Each PDS is keyed using the following format "domain/hub/robot/probe" where the domain, hub, and robot are the target nimbus address and the probe is one of the specified packages for deployment. It is up to the caller to know the target(s) and package(s) they would like to retrieve status for. The parameters specified below "StatusTable" are a part of each individual entry in the table.

Return Value	Type	Description
JobID	String	UUID identifier associated with the job.
JobName	String	Name of the job as specified by the user.
JobDescription	String	Description of the job as provided by the user.
JobStatus	String	Current status of the job. This is selected from the following values: QUEUED, RUNNING, SUCCESS, FAILED, INCOMPLETE
StartTime	Long Integer	Millisecond start time value when the job was added.
EndTime	Long Integer	Millisecond end time value when all the tasks for the job have completed.
StatusTable	PDS	Contains an entry for each task within the job. The key is "/domain/hub/robot/package".

An Entry in “StatusTable”. Each entry is retrieved by key.

Return Value	**Type**	**Description**
JobID	String	UUID of the job that the task is associated with.
TaskId	Integer	An integer identifier for the task.
Status	String	Status of the task. It will be selected from the following: QUEUED, RUNNING, SUCCESS, FAILED, or INCOMPLETE.
Host	String	Target of the distribution. This will be either a hostname/ip or a NimBUS address
Description	String	Description of the error state of the package. It will be empty string if there are no exceptions.
Package	String	Name of the package being deployed.
Version	String	Version of the package being deployed.
Type	String	Specifies the type of job. This is selected from “Robot” or “Probe”.

ADE Archive Sync

Important! This feature is not available in UIM Server 8.0. The proposed API is detailed in this section. Some of the calls exist, but the implementation is incomplete. This feature will be available in UIM Server 8.1.

ADE Archive sync is a redesigned version of the distsrv package forwarding functionality. This functionality is designed with simplicity in mind, but provides a way to setup granular synchronization rules between ADE archives on different hubs. Archive sync is driven by rules created at a “parent” ADE. These rules are used by “child” ADE probes to sync their archives to their assigned parent. The parent-child relationship is established through the callback “set_package_sync_master”

add_package_sync_rule(name,version,rule_type)

Creates a new archive sync rule for this ADE to distribute to its children. When called on a parent ADE probe, any child ADE probes that have called “set_package_sync_master” check to the contents of their archive based on the specified rule. If a child ADE probe(s) does not have a package specified by a rule it will contact the parent ADE probe and download it to archive underneath the child ADE probe.

Input Parameter	Type	Required	Description
name	String	Yes	Name of the package to create a rule for.
version	String	No	Version of the rule to apply the package for. If no verison is supplied then the highest is used.
rule_type	String	Yes	Defines how the sync rule behaves. The following types are allowed: ALL, UPDATE, and SPECIFIC.

delete_package_sync_rule(name)

Deletes the rule for the specific package.

Input Parameter	Type	Required	Description
name	String	Yes	Name of the package to delete a rule for.

list_rules(name)

List all the rules available for this ADE probe.

Input Parameter	Type	Required	Description
name	String	No	Name of the package to list a specific rule for. No name specified means all rules.

The PDS returned by “list_rules” contains an array of PDS entries retrieved using the key “rules”. For the sake of simplicity, the table below covers one of those entries.

Return Value	Type	Description
package	String	Name of the packge the rule applies to
rule_src	String	Nimbus address where the rule was created from.
rule_type	String	Type of rule as specified by the add_package_sync_rule callback.
version	String	Version of the package the rule applies to.

set_package_sync_master(name)

Establishes a parent-child relationship from the called ADE probe to the ADE specified using the “robot” parameter. When this callback is called, the called ADE probe downloads the set of rules that are on the parent ADE probe. It also begins the process of the initial sync with the parent to make sure the contents of parent archive are in sync with the called ADE probe.

Input Parameter	Type	Required	Description
robot	String	Yes	Nimbus address of the hub to which a parent ADE is connected.

unset_package_master_sync

Disables the parent-child relationship between the called child ADE probe and its parent ADE probe. The child ADE probe clears it rule list of all rules downloaded from the parent ADE probe. It does not make any changes to its local archive.

refresh_rules

The called ADE probe attempts to contact its parent ADE probe and download the current rule set. This is useful if there is ever a problem keeping the rules in sync. It does not do any package syncing.